SHARP

SHARP
COMPUTER
SOFTWARE

X68000用

C COMPILER PRO-68K ver2.0

プログラマーズマニュアル

X68000用

C COMPILER PRO-68K ver2.0

プログラマーズマニュアル

SHARP

はじめに

本書は、「C compiler PRO-68K ver2.0」（以下、XC コンパイラと表記します）に含まれている開発ツールの使い方などの解説と、X68000 用のソフトウェアを作成するうえで必要となる技術情報が記載されています。

なお別冊の「C ユーザーズマニュアル」に XC コンパイラの商品構成、動作環境の作成方法、各マニュアルに内容が説明されています。初めて本プログラムをご使用になる方は、必ず「C ユーザーズマニュアル」を先にご覧ください。

MS-DOS は米国マイクロソフト社の商標です。

本書の構成

本書には、開発キットに含まれている開発ツールの使い方などの解説と、X68000 用のソフトウェアを作成するうえで必要となる技術情報が記載されています。
本書は、次の 8 章と付録から構成されています。

第1章　開発ツール

本開発キットに含まれる各開発ツールについて、コマンド、ユーティリティ別に解説しています。ただし、アセンブラ、リンカ、アーカイバ、ライブラリアン、デバッガについては「アセンブラマニュアル」を、ソースコードデバッガについては「ソースコードデバッガ　マニュアル」をご覧ください。

第2章　DOS コール

X68000 で使用できるファンクションコールのうち、Human68k 内部のエントリを呼び出す「DOS コール」を解説しています。

第3章　IOCS コール

X68000 で使用できるファンクションコールのうち、ROM 内部のエントリを呼び出す「IOCS コール」を解説しています。

第4章　日本語 FP のファンクションコール

かな漢字変換システム（日本語 FP）のファンクションコールについて説明しています。

第5章　BASIC 外部関数の作成

X-BASIC の外部関数を作成する方法について説明しています。

第6章　デバイスドライバ

X68000 で動作するデバイスドライバについて、その仕組みや作成例などを説明しています。

第7章　プログラムインターフェイス

ユーザープログラム（ユーザーが作成したプログラム）が Human68k 上で動作するさいのシステムの状態などについて説明しています。

第8章　Human68k ver. 2.0 の新機能

Human68k ver. 2.0 で拡張されたバックグラウンド処理などの機能について、サンプルプログラムつきで説明しています。

付　録

メモリマップ、ディスクマップ、システムファイル、画面構成、SRAM、FDC、IOCS.X などを補足的に説明しています。

開発システムの概要

本開発キットは、X68000のオペレーティングシステムHuman68k上で動作するソフトウェアを作成するための、一連のソフトウェアツールです。
次にこの開発システムの特長を記します。

アセンブリ言語、C言語による開発

この開発キットを使うことによって、アセンブリ言語あるいはC言語によるソフトウェアの作成ができます。
これらのキットを使って作成したソフトウェアは、X-BASICで作成したプログラムがX-BASICインタープリタのもとで機械語に翻訳しながら実行されるのとは異なり、アセンブル／コンパイル、リンクといった手順を踏んで実行形のプログラム（機械語のプログラム）に変換され、Human68kからコマンドの1つとして直接実行させることができます。
アセンブラやコンパイラを使ったソフトウェア作成の利点の一つは、プログラムをいくつかの部分に分けてプログラミングし、別々にコンパイルまたはアセンブルした後、リンカで1つのプログラムにまとめることができることです。
このような方法で開発をすることにより、1度作ったプログラムを後で別のプログラムに組み入れて活用したり、複数人で同時にプログラミングを進めたりすることができます。
また、アセンブラでなくては記述できない部分だけをアセンブラで記述し、あとの部分を開発が容易なC言語で記述してリンカで1つにまとめれば、プログラミング効率をよくすることもできます。

強力なデバッギングツール

開発キットには、機械語およびアセンブリ言語のレベルでプログラムをデバッグできるシンボリックデバッガ（DB）と、C言語のソースレベルでデバッグできるソースコードデバッガ（SCD）が含まれています。
ソースコードデバッガは、従来のDBに機能に加えマウスを使ったフルスクリーンモードを用いて、デバッグ中にC言語のソースコードを参照したり、C言語で宣言された変数を参照や変更することができます。また、トレース、ステップ実行、ブレークポイント、ウォッチポイントなどをC言語レベルで利用できる強力なツールです。

開発に便利なツール群

アセンブリ言語のソースファイルを管理するためのアーカイバや、オブジェクトをまとめて管理するライブラリアン、プロセスの状態をモニターするコマンドなど、開発時に役立つツールが用意してあります。

アセンブリ言語による開発

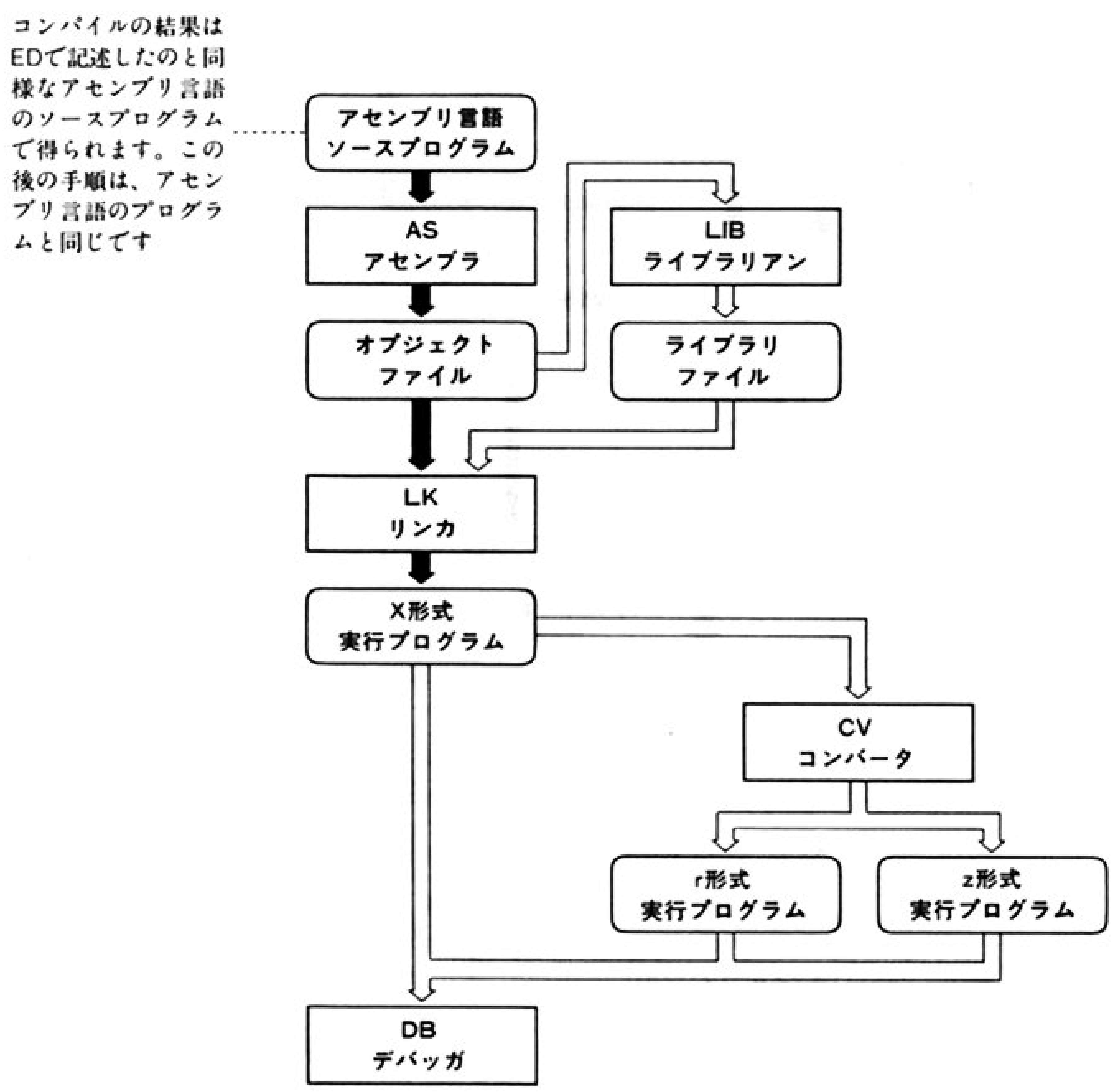
特に高速であることを要求されるソフトウェアを作成するとき、また、ハードウェアを直接に制御するなど、C 言語などの高級言語では扱いにくいソフトウェアを作成するときには、CPU が持っている命令語セットと一対一に対応している機械語によるプログラミングが必要になります。

本開発キットには、X68000 の CPU として使われている MPU68000 用のアセンブラが含まれています。

このアセンブラは、MPU68000 が持っている命令語で記述した（ニーモニックコードで表す）プログラムを MPU68000 が実行できるような形に変換するものです。

さらに、このアセンブラは、繰り返し記述するような手続きをあらかじめ定義しておけるマクロ機能や、メモリ配置に関する疑似命令など、機械語のプログラムを作成するのに便利な機能を持っており、手間がかかる機械語でのプログラミング作業を軽減できるようになっています。

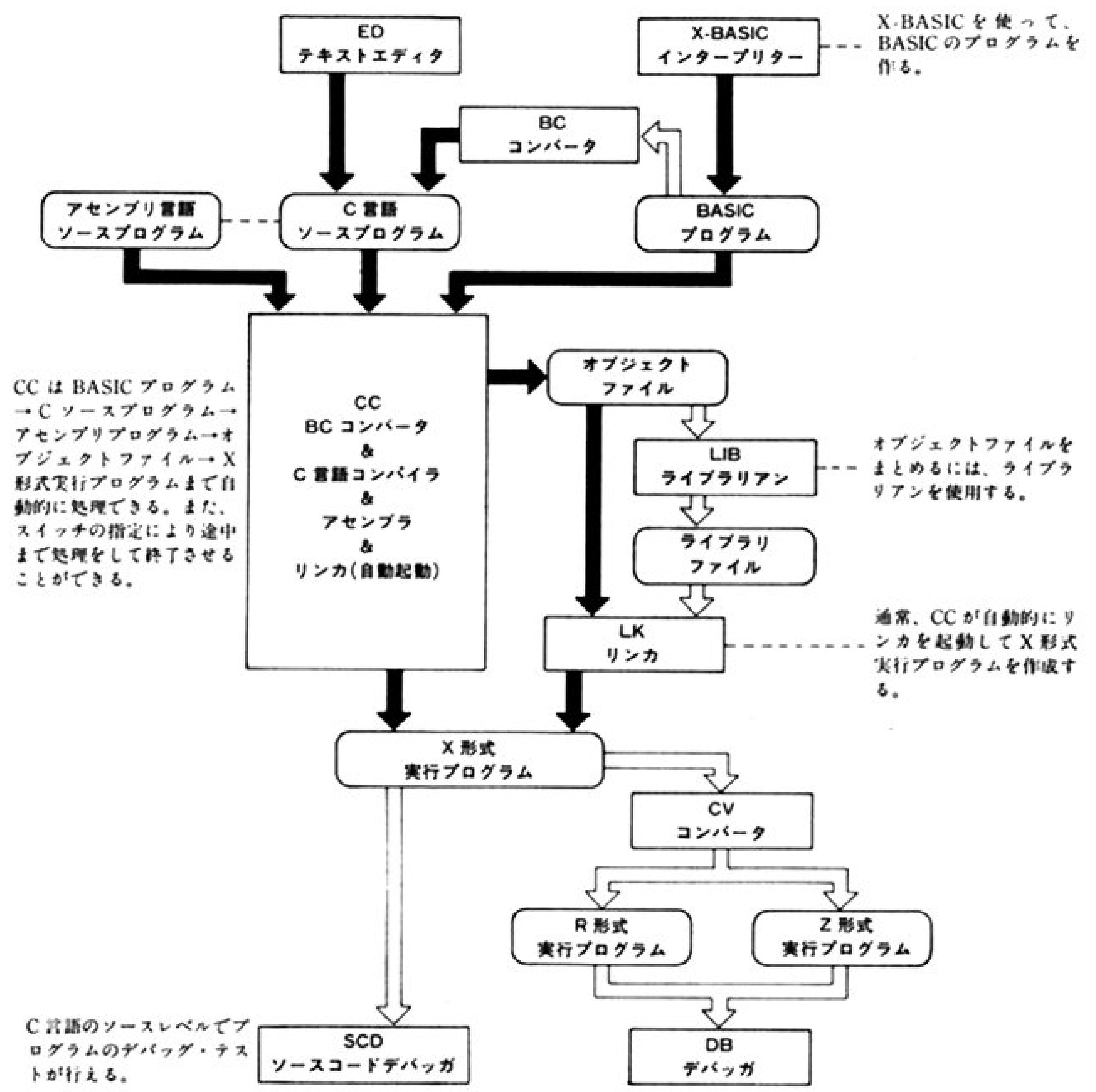
アセンブリ言語による開発の手順を次の図に示します。



〔アセンブリ言語による開発の手順〕

C言語による開発

本開発キットには、C言語で書いたプログラムをX68000上で実行可能なプログラムに変換するためのC言語コンパイラが含まれています。
C言語は、UNIXやMS-DOSで広く普及しているプログラミング言語で、アセンブリ言語に較べてプログラムの記述が容易なこと、他の高級言語に較べてきめの細かい記述ができること、柔軟な構造であること、また異機種間での移植性が高いことなどを特長としてあげることができます。
本開発キットでは、テキストエディタでC言語のプログラムを書いてコンパイルするという一般的な手段のほか、特にX-BASICで作成したプログラムをC言語のプログラムに変換するコンバータ（BC）が用意してあります。
これによって、デバッグが容易なX-BASICでプログラムを作成し、完成後に効率のよいC言語のプログラムへ変換することもできます。



〔C言語による開発の手順〕

CONTENTS

第1章　開発ツール

第2章　DOSコール

第3章　IOCSコール

第4章　日本語FPのファンクションコール

第5章　BASIC外部関数の作成

第6章　デバイスドライバ

第7章　プログラムインターフェイス

第8章　Human68k ver2.0 の新機能

付　録

第1章

開発ツール

外部コマンド

プログラム保守ユーティリティ（MAKE）

本章は、本ソフトウエア開発キットに含まれる開発ツールの簡単なリファレンスマニュアルです。

なお、ここで解説していないアセンブラやコンパイラ、リンカ、ライブラリアン、アーカイバ、ソースコードデバッガなどについての解説は、付属の「アセンブラマニュアル」、「Cユーザーズマニュアル」、「Cリファレンスマニュアル」、「Cライブラリマニュアル」、「ソースコードデバッガマニュアル」をご覧ください。

1.1 外部コマンド

各コマンドの起動は、Human68k のコマンドモードから行ってください。
なお、コマンド別解説では、次のような表記法に従っています。

[ ]	**角形カッコ**	オプション項目です。 省略できます。 必要に応じて入力してください。
< >	**山形カッコ**	入力するデータ名を表しています。
{ }	**波形カッコ**	カッコ内で、"｜" で区切られた項目のうち、必要なものを最低一つ入力することを表します。
………	この記号の直前に書かれている項目を複数個入力できることを表します。 もちろん、一つだけでもかまいません。	

また、本章でコマンド入力の例をあげているときは、"A>"などの Human68k のプロンプトを省略し、キー入力文字を大文字で書いてあります。
もちろん、キー入力文字は小文字でもかまいません。

CASE

機能　ファイル名に含まれる半角英字を、大文字または小文字に変更します。

書式　CASE　[/L | /U]　[/D]　[〈パス名〉 | 〈ファイル名〉]

解説　〈ファイル名〉で指定したファイルの名前に含まれる半角の英文字を、半角小文字または半角大文字に変更します。
ただし、ファイルに読み出し専用属性、システム属性がついている場合は、変更の対象から除外されます。
ファイルの属性については、本体添付の「Human68k ユーザーズマニュアル」の「ATTRIB コマンド」の項を参照してください。
〈ファイル名〉の指定にはワイルドカードが使用できます。
〈ファイル名〉が省略されたときは、カレントドライブ、カレントディレクトリにある、すべてのファイルが対象になります。
スイッチが省略されたときは、大文字に変更します(/U)。

スイッチ

/L
ファイル名の半角英字を小文字に変更します。

/U
ファイル名の半角英字を大文字に変更します(デフォルト)。

/D
ディレクトリ名も変更の対象とします。

例

CASE　/U　*.X ↵
カレントドライブ、カレントディレクトリで、拡張子が".X"であるすべてのファイルのファイル名の半角英字を大文字にします。

CASE　/D　¥BIN ↵
カレントドライブのディレクトリ¥BIN のすべてのファイル名、ディレクトリ名を大文字にします。

COPYALL

機　能　ファイルおよびディレクトリのコピーを行います。

書　式　COPYALL　[／D]　[／T｜／N]　[／V]　[／U]　[／S]
[〈d1：〉]　[〈パス名1〉]
〈ファイル名1〉　{〈d2：〉｜〈パス名2〉}

解　説　[〈d1：〉][〈パス名1〉]〈ファイル名1〉で指定したファイルを{〈d2：〉｜〈パス名2〉}で指定されたパスにコピーします。
このとき、転送先のディスクがいっぱいになった場合、ディスクを交換するようにメッセージを表示し、交換すると引き続きコピーを行います。
COPYALL コマンドは種別の違うディスクのオールコピーもできますし、COPY コマンドと異なり、新しいファイルだけをコピーするといった選択的なコピーやサブディレクトリの内容までコピーできます。
また、BACKUP コマンドがシステム全体のバックアップを作成するために使用するのに対し、COPYALL コマンドは新しく作ったファイルだけをバックアップするときなどに利用できます。

スイッチ

／D

サブディレクトリまでコピーしない場合に指定します。
このスイッチがなければ、サブディレクトリ以下をコピー先にも同じサブディレクトリを作りながらコピーします。

／T

コピー元とコピー先のファイルを比較し、コピー元のファイルが新しいかコピー先に存在しないファイルであればコピーします。このスイッチがなければすべてコピーします。

／N

コピー元とコピー先のファイルを比較し、コピー元のファイルが新しければコピーします。
コピー先に存在しないファイルはコピーしません。

／V

コピー動作の表示のみを行い、実際にはコピーを行いませ,
このスイッチは／T は／N と同時に指定し、コピー元とコピー先でどのファイルが新しくなったかを調べる時に指定します。

／U

コピーするファイル名に英小文字が含まれていた場合、ファイル名をすべて英大文字に変えてコピーします。

／S

コピー元にあるシステムファイルを、コピーしなくてよい時に指定します。
"HUMAN. SYS" などをコピーの対象から外したい時などに使用します。

注：これらのスイッチのうち、／Tと／Nを同時に指定することはできません。

例

COPYALL　A：¥＊.＊　B：⏎

ドライブA（ハードディスク）のすべてのファイルをドライブB（フロッピーディスク）にコピーする

COPYALL　／D　A：¥＊.＊　B：⏎

ドライブAのルートディレクトリのファイルだけをドライブBにコピーする

COPYALL　／T　／U　A：¥＊.＊　B：⏎

ドライブAの新しくなったすべてのファイルを、ドライブBにファイル名を大文字にしてコピーする

COPYALL　／S　／D　A：¥＊.＊　B：⏎

ドライブAのルートディレクトリのシステムファイル以外のファイルを、Bドライブへコピーする

COPYALL　／D　／T　A：¥＊.＊　B：⏎

ドライブAのルートディレクトリで新しくなったファイルだけをドライブBにコピーする

COPYALL　／N　A：¥＊.＊　B：⏎

ドライブBのファイルと比較して、Aドライブの新しいファイルだけをコピーする

DRIVE

機　能　ドライブの種類、状態を表示します。
また、ドライブ名を交換します。

書　式　DRIVE　[／V]　[／D]　[〈d1：〉]　[〈d2：〉]

解　説　DRIVEコマンドは、ドライブ名〈d1：〉と〈d2：〉を指定するかどうかで動作が異なります。

〈d1：〉、〈d2：〉でのドライブ名の指定がない場合は、全ドライブの種別を表示します。

ドライブ名〈d1：〉だけを指定すると、そのドライブの状態を表示します。

ドライブ名を〈d1：〉、〈d2：〉とも指定した場合は、指定したドライブ名を交換します。

スイッチ

／V

ボリュームラベルを付けて全ドライブのタイプ表示を行います。
ハードディスクの領域を分割して利用しているときに、ボリュームラベルを確認するためのものです。

／D

すべてのドライブ名を初期状態に戻します。
ドライブ名の交換を行っていて、初期状態が分からなくなった時などに指定します。

注：スイッチ／Dをつけるときは、〈d1：〉、〈d2：〉は指定しないでください。

例

DRIVE ↵

A：ハード　ディスク　　ユニット番号 ………………　0
B：2HD (1MB)　　ユニット番号 ………………　0
C：2HD (1MB)　　ユニット番号 ………………　1
D：RAM DISK　　ユニット番号 ………………　0

DRIVE　A：↵

A：ハード　ディスク　　ユニット番号 ………………　0
　1セクタあたりのバイト数 ……………………………1024
　1クラスタあたりのセクタ数 …………………………　1
　総クラスタ数 ……………………………………………20155
　ファイルアロケーションの先頭セクタ番号 ………　1
　ファイルアロケーションのセクタ数 ………………　40
　ルートディレクトリの先頭セクタ番号 ……………　81

ルートディレクトリの最大個数 …………………… 512
データ領域の先頭セクタ番号 ……………………… 97
アクセス ……………………………………………… 可
イジェクト …………………………………………… 不可
書き込み ……………………………………………… 可

DRIVE　A：　D：⏎

ドライブA：とドライブD：を入れ替えました
カレントA：をドライブD：に切り替えました

A：RAM DISK	ユニット番号 ………………	0
B：2HD (1MB)	ユニット番号 ………………	0
C：2HD (1MB)	ユニット番号 ………………	1
D：ハード　ディスク	ユニット番号 ………………	0

この例では、DRIVEコマンドの実行前はA：ハードディスク、D：RAMディスクだったのが、実行後はA：RAMディスク、D：ハードディスクとなります。

MOVE

機　能	ファイルを別のディレクトリに移動します。
書　式	MOVE　{〈d：〉｜〈パス名1〉｜〈ファイル名1〉}　[〈パス名2〉]

解　説

〈パス名1〉または〈ファイル名1〉で指定したファイルを、他の〈パス名2〉で指定したディレクトリへ移動します。
移動ですから、移動元のファイルはなくなります。
〈ファイル名1〉にはワイルドカードが使用できます。
ただし、読み出し専用属性、システム属性のついているファイルは移動されません。
また、移動先のディレクトリは前もって作成されていなければなりません。

例

●ファイル単位の移動

MOVE　TEST. TXT　¥TMP ⏎

カレントドライブのカレントディレクトリにあるTEST. TXTを、ディレクトリ¥TMPへ移動します。

MOVE　*.TXT　¥TMP ⏎

カレントドライブのカレントディレクトリにあるすべての"*.TXT"を、カレントドライブのディレクトリ¥TMPへ移動します。

MOVE　¥TMP¥TEST. TXT ⏎

カレントドライブのディレクトリ¥TMPにあるTEST. TXTを、カレントドライブのカレントディレクトリへ移動します。

●ディレクトリ単位の移動

MOVE　¥TMP　¥TEST ⏎

カレントドライブのディレクトリ¥TMPのすべてのファイルを、カレントドライブのディレクトリ¥TESTに移動します。

MOVE　¥TMP ⏎

カレントドライブのディレクトリ¥TMPのすべてのファイルを、カレントドライブのカレントディレクトリに移動します。

PRINT

機　能　バックグラウンドでファイルをプリンタに出力します。

書　式　PRINT　[／W〈n〉]　[／L〈n〉]　[／H〈ヘッダ文字〉｜／T]
[／B〈タブサイズ〉]　[／F]　[／N]　[／K｜／P]　[〈ファイル名〉]
[〈デバイス名〉]

解　説　〈ファイル名〉で、印刷するファイルを指定します。
指定されたファイルは、バックグラウンド処理を行ってプリンタに出力します。
そのときに印刷状況を表示します。
存在しないファイルは指定できません。
〈デバイス名〉で、出力するデバイス名を指定します。
省略すると PRN になります。
デバイス名以外は指定できません。
注：バックグラウンド処理を行ってプリンタへ出力しますので、“CONFIG.SYS”に“PROCESS=〈プログラム数〉〈レベル〉〈タイムスライス〉”(例：PROCESS=5 2 20)を設定しておく必要があります。

スイッチ

／B〈タブサイズ〉
水平タブサイズを指定します。
〈タブサイズ〉には2、4、8、16のうちいずれかを指定します。
デフォルト値は8です。

／F
改ページ時にフォームフィードコードを出力します。
デフォルトはCR／LFコードによって、行スキップします。

／H〈ヘッダ文字〉
ヘッダ文字を指定します。
指定すると各ページの上部にヘッダが付加され、指定しない場合は〈ファイル名〉がヘッダとして付加されます。

／K
すべての印刷を取り消します。

／L〈n〉
1ページの行数を指定します(〈n〉は指定する行数を表します)。
デフォルト値は66です。

／N
行番号を付けて出力します。
これを指定すると、対象ファイルの各行に順に行番号が付けられます。

／P

印刷の中断／再開を行います。

／T

ヘッダ文字を付けないで出力します。

／W <n>

1行の文字数を指定します（<n> は指定する文字数を表します）。

デフォルト値は80です。

すべてを指定せずに実行すると印刷状況のみを表示します。

例

次にPRINTコマンドを実行したときの例を示します。

```
PRINT /L30 /W72 /B4 config.sys↵
状態       デバイス    行   桁    タブ   ファイル名
------ -------- --- --- ---- ----------------------------------
印刷中    PRN         30   72     4   A:¥config.sys
```

それぞれの意味は、以下のとおりです。

状態……………現在の状態を示します

デバイス………出力するデバイス名

行………………1ページの行数

桁………………1行の文字数

タブ……………水平タブのサイズ

ファイル名……印刷されるファイル名

●印刷すべきものが何もないと、次のように表示されます。

PRINT ↵

印刷待ちファイルはありません

PROCESS

機　　能　プロセスの情報を表示します。

書　　式　PROCESS　[／A｜／B]

解　　説　現在のプロセスが、どのようにメモリを割り当てているかを表示します。
また、並行処理を行っているときのバックグラウンドプロセスや、全プロセスの情報を表示します。

スイッチ

／A

バックグラウンドプロセスを含め、すべてのプロセスの情報を表示します。

／B

バックグラウンドプロセスの情報だけを表示します。
なお、バックグラウンドプロセスは、"CONFIG. SYS" に "PROCESS" の指定がないと実行できません。

例

●バックグラウンドプロセスの情報を表示する。

PROCESS　／B ⏎

これを実行すると、たとえば次のように表示されます。

```
------------------------------------------------------------------
PSP ID ID  プログラム名      ﾓｰﾄﾞ ｽﾘｰﾌﾟ  SSP    USP    ﾊﾞｯﾌｧ  長さ
------ -- ----------------- -- ------ ------ ------ ------ ------
0550D0 00 Human68k system   02 000000 059E84 059E84 00DA52 000080
0414B0 01 SYSTEM TIMER V1   -- 0000E7 043252 044A4E 0419BA 000080
```

それぞれの意味は次のとおりです。

PSP ID

バックグラウンドプロセスのプロセス切り替え識別番号。

ID

バックグラウンドプロセスの管理番号。
プロセスの制御はこの番号で指示します。

プログラム名

バックグラウンドプロセスのプログラム名。

モード

実行モード。
数値はタイムスライス何回に1回実行するかを示します（16進数）。

ただし、"--" のときはそのプロセスがスリープ（停止）していることを示します。

スリープ

スリープ残り時間（16進数）。

次回起動されるまでの時間を示します。

ただし、数値が000000でかつモードが "--" のときは、他から起動されるまでスリープしています。

これ以外の場合は、バックグラウンドで実行されていることを示します。

SSP

プロセスのSSPレジスタの値（16進数）。

USP

プロセスのUSPレジスタの値（16進数）。

バッファ

プロセス間通信用バッファの先頭アドレス（16進数）。

長さ

プロセス間通信用バッファの長さ（16進数）。

●全プロセスの情報を表示する。

PROCESS ／A ⏎

これを実行すると、たとえば次のように表示されます。

```
------------------------------------------------------------------
 開始   終了   長さ  モード            ファイル名
------ ------ ------ ------ -----------------------------------
006800 085FFF 07F800 SUPER  Human.sys
086010 0860FF 0000F0 MALLOC
086110 08630F 000200 MALLOC
09A480 09B56F 0010F0 USER   A:¥COMMAND.x
09B580 0A2843 0072C4 MALLOC
0A2860 0A327B 000A1C MALLOC
0A3290 0A348F 000200 MALLOC
0A34A0 0A369F 000200 MALLOC
0A36B0 40BFFF 368950 USER   A:¥BIN¥PROCESS.x

086320 09A1FD 013EDE KEEP   A:¥BIN¥TIMER.x
09A210 09A467 000258 KEEP   A:¥BIN¥VDSP.x
------------------------------------------------------------------
PSP ID ID  プログラム名  ﾓｰﾄﾞ ｽﾘｰﾌﾟ  SSP    USP   ﾊﾞｯﾌｧ  長さ
------ -- -------------- -- ------ ------ ------ ------ ------
0A36B0 00 Human68k system 02 000000 0A8480 0A8480 00E33E 000010
086320 01 SYSTEM TIMER V1 -- 00037B 08832E 089B2A 086A0A 000004
```

上段にはすべてのプロセスの情報が表示され、下段にはバックグラウンドプロセスの情報が表示されます。

上段に表示されたものの意味は次のとおりです。

開始

メモリブロックの開始アドレス（16進数）。

データ部または PSP の先頭アドレスを示します。

終了

メモリブロックの終了アドレス（16進数）。

長さ

メモリブロックの長さ（16進数）。

メモリ管理領域の16バイトは含まないデータ部のみの長さ。

モード

メモリブロックの属性を示します。

属性はそれぞれ以下のような意味です。

SUPER	スーパーバイザモードで実行
USER	ユーザーモードで実行
MALLOC	直前に表示された SUPER、USER、KEEP になっているプロセスによって確保されたメモリ
KEEP	常駐プロセス。 バックグラウンドプロセスも含みます。

ファイル名

起動されたプロセスのパス名とファイル名。

TERM

機能　ターミナルモードに入ります。

書式　TERM　[〈ファイル名〉]

解説　ターミナルモードに入ります。

漢字コードはシフト JIS コード、改行は CR／LF を使用します。

なお、ボーレート、ビット長などは、「SPEED コマンド」(RS-232C インターフェイスに対するパラメータの設定・起動）での設定が使われます。

〈ファイル名〉は、以下で解説するファンクションキーの[F8]で使用します。

なお、ターミナルモードでは、本体添付の「Human68k ユーザーズマニュアル」にあるエスケープシーケンスが有効です。

ファンクションキーに以下のコマンドが設定されています。

[F6]　Full／Half

通信方法を切り替えます。

Full：全二重

Half：半二重

[F7]　PrnOff／PrnOn

画面表示内容をプリンタに出力するかどうかを切り替えます。

PrnOff：プリンタに出力しない

PrnOn：プリンタに出力する

[F8]　WrtOff／WrtOn

画面表示内容をファイルに保存するかどうかを切り替えます。

コマンドラインで〈ファイル名〉を指定すると、その名前のファイルが作成されます。

無指定のときは"TERM. $$$"という名前で作成されます。

WrtOff：ファイルに保存しない

WrtOn：ファイルに保存する

なお、ファイル保存している最中は、画面のファンクションキー表示部が WrtOn に変わります。

また、ターミナルモードでは何度 WrtOn／WrtOff を繰り返してもかまいません。

このような場合は、ファイルには WrtOn になっているときの画面がどんどん追加されていきます。

F9 Command

OSのコマンドを実行します。

F10 End

ターミナルモードを終了します。

例

TERM SCREEN. DOC ⏎

F8でWrtOnにしたときに画面表示内容をSCREEN. DOCに保存することにして、ターミナルモードに入ります。

TOUCH

機　能　ファイル、ボリューム名のタイムスタンプの時刻、日付を変更します。

書　式　TOUCH　[／T 〈hh〉：〈mm〉：〈ss〉]　[／D 〈yyyy〉—〈mm〉—〈dd〉]
[／V｜／S｜／A]　〈ファイル名〉

解　説　ファイルやボリューム名のタイムスタンプを変更します。
〈ファイル名〉にはワイルドカードが使用できます。
ただし、読み出し専用属性、システム属性のついているファイルは変更の対象から除外されます。

スイッチ

／T 〈hh〉：〈mm〉：〈ss〉

ファイルの作成時刻を指定時刻に変更します。
秒指定は偶数秒単位でなければなりません。
奇数秒を指定した場合、偶数秒に補正されます。

hh　：時（0～23）
mm：分（0～59）
ss　：秒（0～58までの偶数）

／D 〈yyyy〉—〈mm〉—〈dd〉

ファイルの作成日付を指定日付に変更します。

yyyy：年（1980～2079または80～99、00～79。2桁の場合80～99は1980～1999とみなし、00～79は2000～2079とみなされます）
mm：月（1～12）
dd　：日（1～31）

／V

ボューム名の作成時刻、日付を変更します。

／S

サブディレクトリの作成時刻、日付を変更します。

／A

ファイル、ボリューム名、サブディレクトリのすべての作成時刻、日付を変更します。

注：スイッチ／T、／Dを指定しない場合は、現在の日時を設定します。

例

TOUCH　TEST. TXT ⏎

ファイルTEST. TXTのタイムスタンプを、現在の日時に変更します。

TOUCH　＊.＊⏎

カレントドライブ、カレントディレクトリのすべてのファイルのタイムスタンプを、現在の日時に変更します。

TOUCH　/T12:00:00　TEST. TXT ⏎

ファイル TEST.TXT のタイムスタンプの日時を現在の日付の 12 時 00 分 00 秒に設定します。

TOUCH　/D90—07—23　/T12:10:20　TEST. TXT ⏎

ファイル TEST. TXT のタイムスタンプの日付と時刻を、90 年 7 月 23 日、12 時 10 分 20 秒に変更します。

TREE

機　能　ディレクトリ構造をツリー形式で表示します。

書　式　TREE　[／F]　〈パス名〉

解　説　〈パス名〉で指定したディレクトリ下のディレクトリ構造を、ツリー形式で表示します。

〈パス名〉が省略された場合は、カレントドライブのルートディレクトリ以下が表示されます。

なお、システム属性、不可視属性のついたディレクトリ、ファイルもすべて表示します。

スイッチ　／F

ディレクトリのツリー構造を表示するとともに、そのディレクトリに属するファイルの名前も表示します。

例

TREE ↵

カレントドライブの、ルートディレクトリ以下のディレクトリ構造がツリー表示されます。

TREE　¥TMP ↵

カレントドライブの、ディレクトリ¥TMP以下のディレクトリ構造がツリー表示されます。

TREE　／F ↵

カレントドライブの、ルートディレクトリ以下のディレクトリ構造がツリー表示されます。

ディレクトリに属するファイルの名前もすべて表示します。

```
¥
|-XCシステム#1
|-HUMAN.SYS
|-CONFIG.SYS
|-KEY.SYS
|-USKCG.SYS
|-BEEP.SYS
|-STARTUP.ENV
|-AUTOEXEC.BAT
|-COMMAND.X
|
|-SYS
|    |-PRNDRV.SYS
|    |-PCMDRV.SYS
|    |-ASK68K.SYS
|    |-RAMDISK.SYS
|    |-SRAMDISK.SYS
|    |-PRNDRV1.SYS
|    |-PRNDRV2.SYS
|    |-PRNDRV3.SYS
|    |-SCSIDRV.SYS
|    |-OPMDRV2.X
|    |-FLOAT1.X
|    |-FLOAT2.X
|    |-HISTORY.X
|    |-IOCS.X
|
|-BIN
|    |-AS.X
|    |-LK.X
|    |-DB.X
|    |-SCD.X
|    |-SCD.CNF
|    |-SCD.HLP
|    |-AR.X
|    |-LIB.X
|    |-CV.X
|    |-BIND.X
|    |-CASE.X
|    |-DRIVE.X
|    |-MOVE.X
|    |-PROCESS.X
|    |-COPYALL.X
|    |-TERM.X
|    |-TOUCH.X
|    |-TREE.X
|    |-WHERE.X
|    |-FORMAT.X
|    |-DISKCOPY.X
|    |-MAKE.X
|    |-PRINT.X
|    |-ED.X
|    |-ED.HLP
|
|-ASK
|    |-ENV1.ASK
|    |-ENV2.ASK
|    |-ENV3.ASK
|    |-ENV4.ASK
|    |-ENV5.ASK
|
|-HIS
|    |-KEY.HIS
|    |-HISTORY.HIS
|    |-HISTORY.HLP
|
|-ETC
|    |-INSTALL.BAT
|    |-FD.BAT
|    |-FDR.BAT
|    |-HD.BAT
|    |-YN
|    |-TOOL.X
```

WHERE

機　　能　ファイルを検索します。

書　　式　WHERE　[／A]　[／F]　{〈d：〉|〈パス名〉|〈ファイル名〉}

解　　説　〈パス名〉または〈ファイル名〉で指定したファイルを、ディレクトリのツリー構造の中から検索します。

〈d：〉で指定したドライブの〈パス名〉で指定したディレクトリから始め、〈ファイル名〉で指定されたファイルをディレクトリのツリー構造に従って順に検索し、見つかったファイルの位置するディレクトリのパス名を表示します。

〈d：〉のドライブ名が省略された場合にはカレントドライブが、〈パス名〉が省略された場合にはルートディレクトリが指定されたものとみなされます。

〈ファイル名〉にはワイルドカードが使用できます。

また、システム属性、不可視属性のついたディレクトリ、ファイルもすべて検索します。

スイッチ

／A

すべてのドライブを検索します。

／F

パス名のほかに、見つかったファイルの名前も表示します。

注：スイッチ／Aをつけるときは、〈d：〉は指定しないでください。

例

WHERE　TEST. TXT ⏎

カレントドライブのルートディレクトリから、ファイル TEST. TXT を検索し、その位置をパス名で表示します。

WHERE　／F　*. TXT ⏎

カレントドライブのルートディレクトリから、拡張子が". TXT"であるファイルを検索し、その位置をパス名とファイル名で表示します。

WHERE　／A　TEST. TXT ⏎

すべてのドライブのルートディレクトリから、ファイル TEST. TXT を検索し、その位置をパス名で表示します。

1.2 プログラム保守ユーティリティ　MAKE

1.2.1 はじめに

多くのソースプログラムからなるプログラムの開発では、コンパイル、アセンブル、リンク、その他の作業を一つずつキーボードからコマンドで実行するとたいへんな数になります。
「そういう面倒なことはプログラム化しよう」と考えた時、まず、思い付くのはバッチファイルでしょう。
しかし、たとえば、最初にそれを走らせてコンパイル中にエラーが出たとします。
そうすると、エラーのでたファイルのデバッグ後、もう1度バッチプログラムを実行します。
さて、バッチプログラムはコンパイルの必要／不必要に関わらず、全てのファイルを再コンパイルしてしまいます。
これでは、ファイルの数が少ない時はまだ良いのですが、多くなると開発効率が極端に落ちてしまいます。
こういう、複数のソースプログラムからなるプログラム開発に有効なのが、MAKEです。

MAKEはアセンブラやコンパイラによるプログラム開発の保守、更新、再生成を自動的に行うプログラム保守ユーティリティです。
MAKEコマンドを実行すると、あらかじめユーザーが作ったメイクファイルを読み込みます。
メイクファイルには、作成する目的のファイルがどんなソースプログラムから構成されているかということ（依存関係）と、その依存するファイルを用いて目的のファイルを作成するために実際にDOS上で実行されるコマンドが書かれています。
これ以降、作成するファイルをターゲットファイル、依存するファイルを依存ファイルということにします。

MAKEはターゲットファイルと、その依存ファイルの作成日時を比較し、

依存ファイルの作成日時が新しい時だけ、ターゲットファイルの更新処理を自動的に行います。
つまり、ターゲットファイルを更新するために必要最小限の処理を自分で判断して実行してくれるのです。

MAKEは複数のファイルからなるプログラム開発で特に力を発揮します。
しかし、1本のソースプログラムからできているものでも、メイクファイルを作成して、将来の拡張やデバッグに備えることを、お勧めします。
1.2.2では最も基本的な使い方を、1.2.3では具体的なMAKEの使い方を、1.2.4ではMAKEを活用するための各機能の詳細を説明します。

1.2.2 初めてMAKEを使う方に

ここでは、MAKEを使うのに必要最小限のことを説明します。
とりあえず、使ってみてMAKEの便利さを感じてみてください。

●手順

1) アセンブラやC言語などのソースプログラムを作る。
　MAKEはプログラムの開発を支援するものです。
　まず、プログラムを作ってみましょう。
2) メイクファイルを作成します。
　ED.Xなどのエディタを使って、メイクファイルを作成します。
　メイクファイルには、ターゲットファイルを作成するために必要な事項が書かれています。
　詳しくは、「メイクファイルの構成」を参照してください。
3) MAKEコマンドを実行します。
　「MAKEの起動」を参照してください。
4) エラーが出たら、デバッグして3)を行います。
　エラーが無くなったら完成です。

●メイクファイルの構成

メイクファイルの基本の書式は次のようになります。

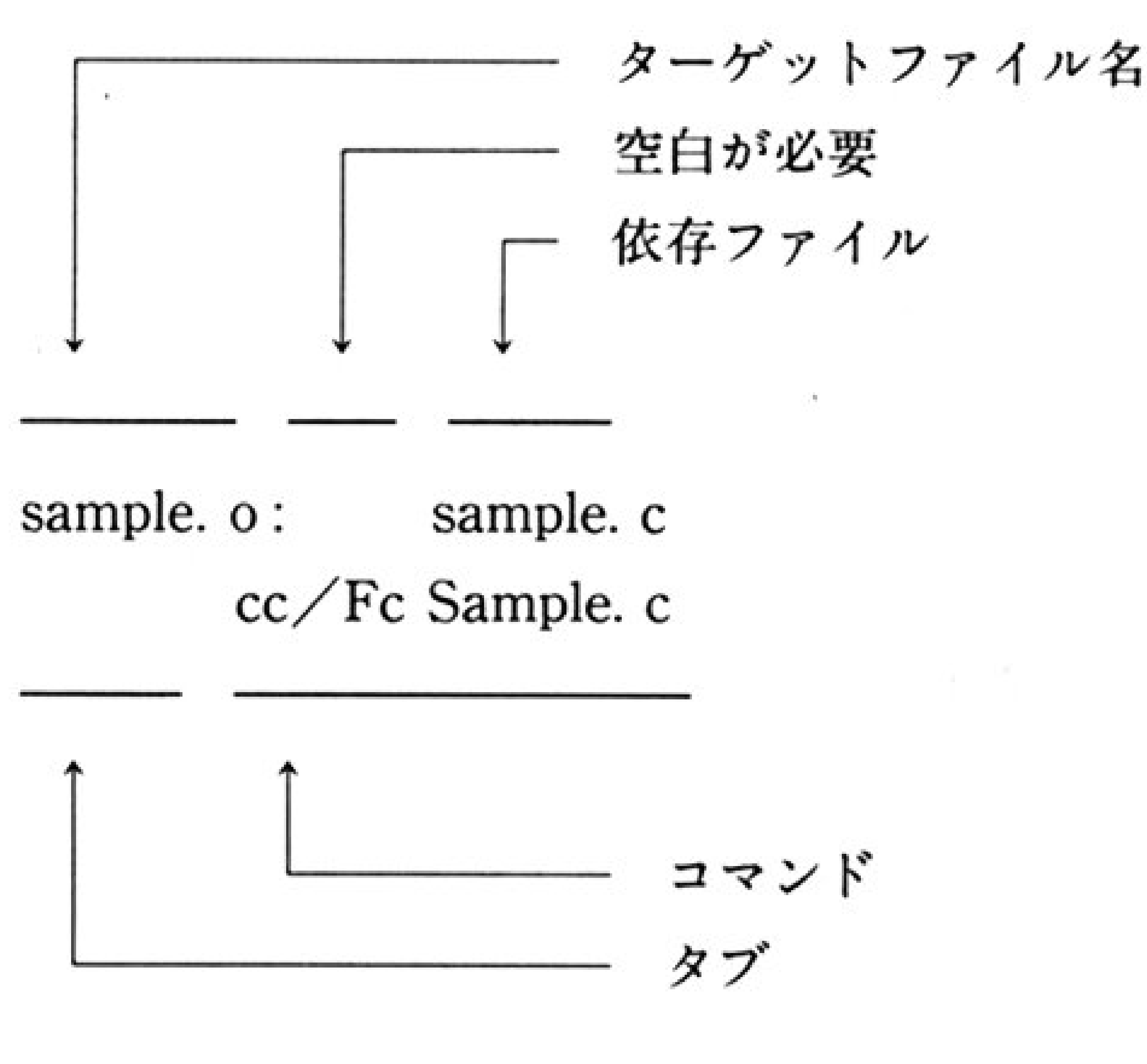
1.2 プログラム保守ユーティリティ MAKE

```
ターゲットファイル名：依存ファイル名・・・
〈タブ〉コマンド
              ・
              ・
              ・
```

ターゲットファイル	：	実際に作成するファイルで、その形式(".X"や".O"など)のファイル
依存ファイル	：	ターゲットを構成するファイル名
コマンド	：	ターゲットファイルを更新するためにDOS上で実際に行うコマンド
タブ	：	[TAB]キーを押すと入力されるキーコード

メイクファイルは、2種類のリストを組み合わせたテキストファイルです。上段に作りたい目的のファイル名を書き、'：'で区切ってそのファイルを構成するファイル名を並べて書きます。
依存ファイルは空白で区切り、下段に、タブに続けてターゲットを作成するDOSコマンドを書きます。
コマンドはいくつ書いてもかまいません。
メイクファイルでは最初に最終的なターゲットファイルを書きます。
なぜなら、MAKEのコマンドラインでターゲットを指定しないと、最初に見つけられたファイルがターゲットとなるからです。
例1に各要素の例を、例2にメイクファイルの例を示します。

［例1］



［例2］ Cとアセンブラとのリンク

```
#最終的にできるファイル
sample.x :  sample.o sub.o
        lk sample.o sub.o
#依存ファイルはターゲットにもなる
sample.o:  sample.c
        cc ／Fc sample.c
sub.o :  sub.s
        as sub.s
```

● MAKEの起動

MAKEを起動する最も簡単な方法は、単に、

MAKE ⏎

と入力すれば良く、そうすると、デフォルトファイル名である"makefile"を読み込み処理を行います。
ただし、メイクファイル名を"makefile"（大/小文字の区別なし）にしておく必要があります。
基本的には次の書式でコマンドを実行します。

```
MAKE 〈スイッチ〉〈ターゲット〉…
```

スイッチ：1.2.4 各機能の詳細説明の「● MAKEコマンド・スイッチリファレンス　P55」を参照してください。
ターゲット：作成するファイル名です。

メイクファイル名を別のファイル名にするなら、／fスイッチを使うことによって、任意のメイクファイルを指定できます。

1.2.3 メイクファイルのサンプル

ここでは、メイクファイルの例を具体的にあげ、よく使われる機能の説明を行います。

●マクロを使った例／C とアセンブラの分割コンパイル

リスト1

```
#マクロ定義
CC = cc                        #Cコンパイラ
CFLAGS = /O /L                 #Cコンパイラオプション
AS = as                        #アセンブラ
AFLAGS =                       #アセンブラオプション
LC = lk                        #リンカ
LFLAGS =                       #リンカオプション
PROG  = sample.x               #ターゲット名
CFILE = $(PROG:x = c)          #Cのソースファイル
COBJ  = $(PROG:x = o)          #Cのオブジェクト
CINC  = sample.h sub.h         #Cインクルードファイル
AOBJS = sub.o                  #アセンブラオブジェクト
AFILE = $(AOBJS:o = s)         #アセンブラのソースファイル
AINC  = doscall.mac iocscall.mac
                               #アセンブラインクルードファイル
#規則
$(PROG): $(COBJ) $(AOBJ)                #sample.x: sample.o sub.o
        $(LC) $(LFLAGS) $< $(AOBJ)      #        lk sample.o sub.o       と同じ

$(COBJ): $(CFILE) $(CINC)      #sample.o: sample.c sample.h sub.h
        $(CC) $(CFLAGS) $<     #        cc /O /L sample.c       と同じ

$(AOBJ): $(AFILE) $(AINC)      #sub.o: sub.s doscall.mac iocscall.mac
        $(AS) $(AFLAGS) $<     #        as sub.s                と同じ
```

この例はCで書かれたプログラムとアセンブラで書かれたプログラムを1つにまとめるものです。
この例の特長はマクロを使っていることです。
マクロは、任意の文字列を、別の文字列に置き換えるもので、繰り返し同じ文字列を使うときに有効です。
詳しくは「●マクロ　P30」を参照してください。
以下に、この例について説明します。

1,16：　先頭に'#'がある行は注釈文になります。

2~7：　Cコンパイラ、アセンブラ、リンカの名前とオプションを定義します。
この中で、"CC" や "CFLAG" などはこれらの意味で慣用的によく使われています。
このMAKEでは、それらは単にマクロ名の1つですが、他のMAKE（例えば、OS-9）では特別な意味があるので、気をつけてください。

8~14：　MAKEで使われる各ファイル名を定義しています。
9、10、13行の"$（マクロ名）"は、マクロに定義されている文字列を展開するもので、'：'のあとの等式は、マクロの末尾の文字列を変換するものです。

［例］CFILE＝sample. c
　　　COBJ＝$（CFILE：c＝o）→＝sample. o

17~24：MAKEの依存記憶とコマンドです。
ここでも、"$~" のマクロ呼出を多用しています。
'<'はMAKEが作業中に自動的に設定するマクロで自動マクロと呼ばれています。
'<'は最初の依存ファイルを表します。

●明示的なパターン規則の例

リスト2

```
17:$(PROG): %.x: %.o $(AOBJ)
18:        $(LC) $(LFLAGS) $< $(AOBJ)
19:
20:%.o %.c $(CINC)
21:        $(CC) $(CFLAGS) $<
22:
23:%.o: %.s $(AINC)
24:        $(AS) $(AFLAGS) $<
```

この例は前例の規則を「明示的なパターン規則」で書き変えたものです。
詳しくは、「●明示的なパターン規則　P46」を参照してください。
この結果、9、10、13 行は不要になります。
明示的なパターン規則は、ターゲットと依存ファイルの間にパターンを入れます。
パターンは '%' を含む文字列で、後ろを ':' で区切ります。
MAKE はこれをターゲットと比較し、'%' に一致する部分をそのまま '%' に定義します。

●暗黙のパターン規則の例 1

リスト3

```
17:%.x: %.o $(AOBJ)
18:        @$(LC) $(LFLAGS) $< $(AOBJ)
19:
20:%.o %.c $(CINC)
21:        @$(CC) $(CFLAGS) $<
22:
23:%.o: %.s $(AINC)
24:        @$(AS) $(AFLAGS) $<
```

さらに、リスト 2 を「暗黙のパターン規則」に書き換えたものです。
詳しくは、「●暗黙のパターン規則　P46」を参照してください。

これは、リスト1を一般化したものと考えてよく、このままではターゲットファイルを特定できません。
そこで、コマンド行から

A>MAKE　sample. x ⏎

と入力してターゲットを指定すると、暗黙のパターン規則が適用されてリスト1と同じ動作を行います。
この結果、8行は不要となります。
18、21、24行ではコマンドの先頭に@がついていますが、これをつけるとコマンド行を画面に出力しなくなります。

●暗黙のパターン規則の例2

リスト4

```
 1:CC = cc
 2:CFLAGS = /O /Fc
 3:
 4:AS = as
 5:AFLAGS =
 6:
 7:OBJS = test1.o test2.o test3.o test4.o ¥
 8:        test5.o test6.o test7.o
 9:
10:newlib.l: $(OBJS)
11:        lib /u newlib.l $(OBJS)
12:
13:%.o %.c
14:       $(CC) $(CFLAGS) $<
15:%.o: %.s
16:       $(AS) $(AFLAGS) $<
```

この例は暗黙のパターン規則を使って、カレントディレクトリ上の test?. c をコンパイルし、TEST?. Sをアセンブルしてオブジェクトを作り、それらをまとめて1つのライブラリファイルを作るメイクファイルです。
?には1から7までの数字のどれかが入ります。
以下、この例について説明します。

1~2：Cコンパイラとスイッチのマクロ定義
4~5：アセンブラとスイッチのマクロ定義
7~8：ターゲットの依存ファイルのマクロ定義
10~11：ターゲットファイルの規則
13~16：依存ファイル作成の規則

13～16 行の暗黙のパターン規則では、ターゲットを特定していないため、暗黙の規則（MAKE がターゲットおよび依存ファイルを推測する規則）が働いて、適当なＣのソースファイルおよびアセンブラファイルを探します。ここでは終端ターゲット newlib. l の依存ファイルである test1～7 .o をターゲットとして処理を行います。

1.2.4 各機能の詳細説明

MAKE は単なるユーティリティですが、その機能はちょっとした言語といってもいいほど多彩です。
ここでは、その豊富な機能及びそれに関する注意点を説明します。
よく読んで活用してください。

●マクロ

メイクファイル中で、同じコマンド、スイッチ、ファイル名を繰り返し使うことがあります。そのような時、マクロは有効な道具となります。
マクロとは任意の文字列を表す名前です。
そのマクロ名に文字列を設定すると、MAKE はマクロ名を見つけるとマクロを展開して、もとの文字列に置きかえます。

●マクロ定義

マクロは以下によって定義されます。

- ・MAKE があらかじめ定義
- ・メイクファイルの規則中のマクロ定義
- ・環境変数
- ・メイクファイル中のマクロ定義
- ・コマンド行のマクロ定義
- ・MAKE が処理の状態に応じて自動的に定義

定義されたマクロのうちアルファベット、数字、‘_’で始まるものが集められ、新しい環境に設定されます。

●マクロのレベル

マクロにはレベルがあって、それぞれに強さを持っており、同じ名前のマクロが再び定義されると強い方の定義が残ります。
また、強さが同じならば新しい定義が残ります。
このとき、マクロの再定義はエラーではありません。
以下にマクロの定義レベルを示します。
最初の数字がレベルを示します。
なお、数字が大きい方が強くなっています。

1)デフォルトマクロが設定されるレベル
2)／eスイッチを指定していないとき、環境変数が設定されるレベル
3)メイクファイル中のマクロ定義が設定されるレベル
4)／eスイッチを指定したとき、環境変数が設定されるレベル
5)コマンド行でのマクロ定義が設定されるレベル
6)override 指令での定義が設定されるレベル
7)MAKE が自動的に定義する自動マクロのレベル

●デフォルトマクロ

$	'$'が設定されている
#	'#'が設定されている

●自動マクロ

以下の5つのマクロは、ターゲットの更新中にMAKEの内部で自動的に設定されます。

@	更新中のターゲット名
<	更新中のターゲットが依存しているもののうち、最初のものの名前。他のMAKEでの定義「推論によって導き出されたソースファイルの名前」とは違っていますが、暗黙の規則が適用された時、新たに発生した「ターゲットが依存しているもの」は、ターゲットが依存しているもののリストの先頭に加えられますので、それを呼び出すのに、他のMAKEと同様に'$ <'を使用する事ができます。

*　　ターゲットのステム

ステムとは、明示的なパターン規則で、あるいは暗黙のパターン規則が適用されたときに'%'が適合した部分です。

パターン規則が適用されなかったときには、空文字列となります。

他の MAKE での定義とは違っていますが、少なくとも暗黙の接尾辞規則では結果は同一になります。

?　　更新中のターゲットが依存しているもののうち、ターゲットよりも新しいものの名前のリスト

^　　更新中のターゲットが依存しているものの名前のリスト

これらのうち、'@'、'<'、'*'には'D'と'F'による修飾が可能です。

［例］　$(@D)　更新中のターゲット名のディレクトリ部分。
もしターゲット名がディレクトリを含んでいなければ'.¥'となります。

$(@F)　更新中のターゲット名のファイル名部分。

●特殊マクロ

MAKEOVERRIDES …コマンドに指定されたパラメータのなかで、マクロ定義が取り出されて定義されます。
定義レベルは 4 です。
設定される値は、場合によってクォーティングされます（'や"でくくられます）。
コマンド行にマクロ定義が無ければ、このマクロは新たに定義されません。

MAKE ………………… MAKE が自分自身のパス名を設定します。
定義レベルは 2 です。
もし、"MAKEOVERRIDES"を新たに設定した場合には、それも値に追加されます。
MAKE から MAKE を実行する場合のコマンドラインには、"$(MAKE)" と記述してください。

MAKEFLAGS………… MAKE へのスイッチを設定しておきます。
MAKE はまず MAKEFLAGS を処理してからコマンド行のスイッチを処理します。
MAKEFLAGS は左から順に処理されます。
MAKEFLAGS に設定可能なスイッチは、a, e, i, k, n, q, r, s, S, t, w のみです。
これ以外の文字があっても無視されます。
'／'や'－'は必要ありません。
ただし、あってもエラーにはなりません。
MAKEFLAGS で指定された k スイッチは、コマンド行に／S スイッチを指定することによって取り消すことができます。
MAKE はスイッチを処理した後、MAKEFLAGS を新たに再定義します。
定義レベルは２です。
設定される値は、スイッチの指定状況に応じて、a, e, i, k, n, q, r, s, t, v, w の文字が設定されます。
この新たな MAKEFLAGS は前述した通り新たな環境に設定されますので、再帰的に MAKE を実行すると子の MAKE が解釈することになります。
すなわち、MAKE が再帰的に起動されるような場合、コマンド行で指定したスイッチのうち、上記のものに付いては環境変数を通して伝えられます。
MAKEFLAGS の目的は、この為にあります。
いつも同じスイッチを指定するといって、それを環境変数 MAKEFLAGS に設定しておく事は勧められません。
なぜならば、異なった環境で作業をする必要が発生した場合、期待通りの動作をしなくなってしまいます。

MAKELEVEL………… MAKE は最初にこのマクロを探します。
もしあれば設定されている自分のレベルとし、なければ自分のレベルを０とします。
次に MAKE は自分のレベルに１を加えた値を、新たに環境変数 MAKELEVEL に設定します。
子の MAKE は環境変数で定義されたマクロにより

自分のレベルを知る事ができます。
ユーザーが参照するのはかまいませんが、値を定義してはいけません。

MAKEFILES …………パス名のリストを設定しておくと、MAKEはあらかじめそれをメイクファイルとして読み込みます。
もし設定されているファイルが見つからなかった時、パス名が絶対パス名でなければ／Iスイッチで作られたパスを順次パス名の前に加えて検索します。
それでも見つからなければ無視します。
リストは空白で区切るものとします。
','および';'で区切ってもかまいません。
MAKEFILESが設定されていても、／fスイッチの指定は有効になり、／fスイッチが一切指定されていなくても'MAKEFILES'を読み込むことにかわりはありません。
MAKEFILESによるファイルの読み込みは、メイクファイルを読み込む前に行われるので、メイクファイルでMAKEFILESを定義しても効果はありません。(ただしそれは新たな環境となるので、子のMAKEにとっては意味があります。)

VPATH…………………マクロVPATHに設定されているディレクトリのリストは、ファイルの検索パスとして内部のデータベースに登録されます。
このとき適合パターンは'*'となります。すなわち、どのファイルにも適用されます。
ディレクトリは';'で区切ります。

●マクロ呼び出し

マクロの内容を呼び出したいときは'$'を使います。
たとえば、マクロ'M'の値が欲しければ'$M'と書きます。
マクロの名前の長さが2文字以上ならば"$(MACRO)"または"${MACRO}"と書きます。
()と{}とには機能的な違いはありません。

1.2　プログラム保守ユーティリティ MAKE

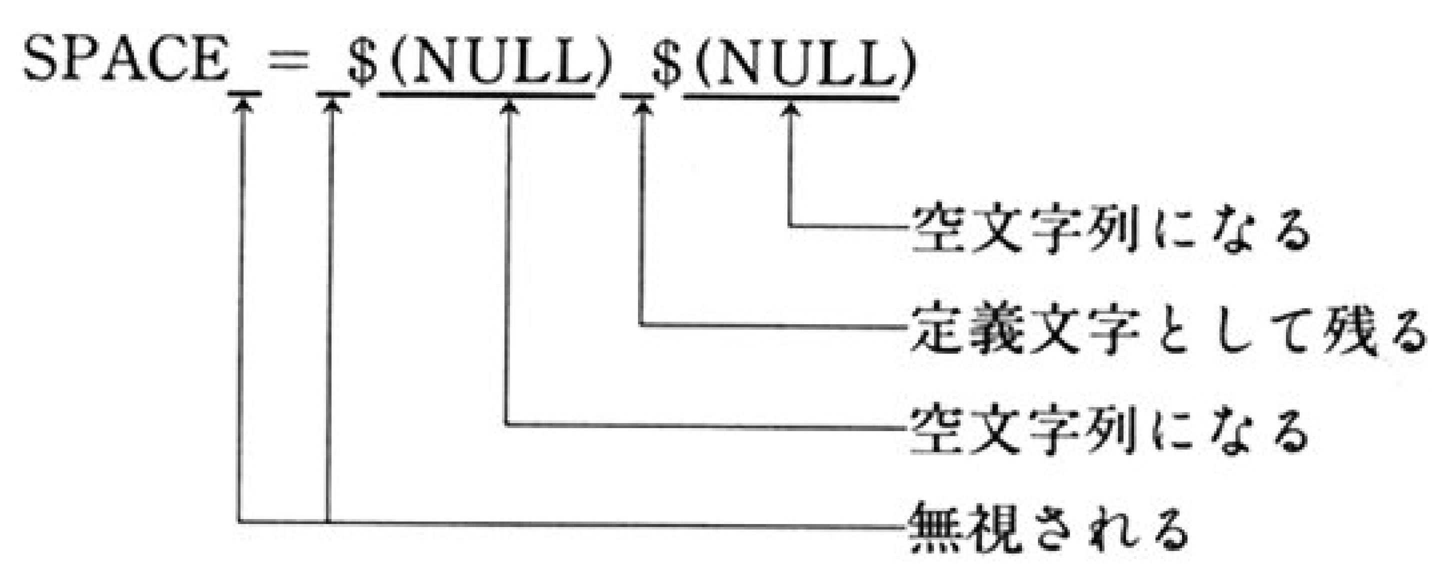
またマクロ呼び出しは、前述した普通の呼び出しに加えて、次の2つの特殊な呼び出し形式が用意されています。

$(#〈名前〉)…………マクロ〈名前〉の値の、空白で区切られた単語の数を出力します。
これは、$(〈名前〉|words)と似ていますが、マクロ〈名前〉の値にマクロ呼び出しが含まれていても、それを展開しません。

$(?〈名前〉)……………マクロ〈名前〉が定義されていれば1を出力し、未定義ならば0を出力します。
これは、
$(calculate $(origin〈名前〉)！= undefined)
と同等です。

マクロ呼び出しにおける注意として、定義されていないマクロを呼び出すとエラーにならず、空文字列になることがあります。
これを利用して、以下のように空白文字などを定義することができます。

[例]



この例では、'='の前後や行末の空白文字は無視されますが、その他の空白文字が無視されないことと、未定義のマクロが空白文字列になることを利用して、マクロ'SPACE'に、空白文字を定義しています。
マクロ定義では、文字列の前後の空白は削除されます。
そこで空白で始まる、あるいは、空白で終わる文字列を定義したい場合、前述したマクロ'SPACE'を利用することで実現できます。
つまり、

MACRO = $(SPACE)terminated by space$(SPACE)
と定義すればよいのです。

マクロの呼び出しでは "$($(MACRO))" のように入れ子にすることができます。

[例]　$(MACRO : search = replace)

この例ではMACROの値を得てから、定義されている内容の語末の（すなわち、空白が続く、あるいは文字列の最後の）"search" を "replace" に置き換えます。

[例]　SRC = file001. s
　　　OBJ = $(SRC : s = o)

この例では、マクロ SRC には、"file001.s" が、マクロ OBJ には、"file001.o" が定義されます。
マクロ呼び出しのパラメータ（<名前> など）の前後に空白がある場合、それは保存されます。
ただし、コマンド名に続く空白は削除されます。
マクロ呼び出しの中の区切り文字','や'='をエスケープする方法は存在しないので、もし、区切り文字をパラメータの一部に使用したい場合は、マクロ展開を利用します。

●マクロコマンド

マクロコマンドもマクロ呼び出しと同じような形をしています。
これを、マクロコマンドの呼び出しといいます。
マクロコマンドには次の5つがあります。

calculate
foreach
join
origin
wildcard

これらを使う時は、小文字でなければいけません。

$(origin 〈マクロ名〉)
〈マクロ名〉が定義されたレベルによって以下の文字列となります。

0) undefined
1) default
2) environment
3) file
4) environment override
5) command line
6) override
7) automatic

左の数値は「マクロのレベル」に対応しています。
ただし、0は未定義を表しています。
〈マクロ名〉がマクロ呼び出しを含んでいるなら、それは事前に展開されます。

$(foreach 〈仮マクロ名〉, 〈リスト〉, 〈文字列〉)
〈リスト〉を空白で区切り、それを1つずつ〈仮マクロ名〉に定義した上で〈文字列〉を展開し、その結果を1文字の空白をはさんで結合します。
〈仮マクロ名〉および〈リスト〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

［例］　$(foreach FILE, test1 test2 test3 test4, a:¥obj¥$(FILE).o)

展開された結果は以下の様になります。

a:¥obj¥test1. o a:¥obj¥tent2. o a:¥obj¥test3. o a:¥odj
¥test4. o

$(join 〈文字列1〉 [〈文字列2〉])
〈文字列1〉と〈文字列2〉を空白で区切り、それを順に結合します。
〈文字列2〉がなければ、単に〈文字列1〉が出力されます。
ただし、空白は保存されません。
〈文字列1〉および〈文字列2〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

$(wildcard 〈リスト〉)

〈リスト〉を空白で区切り、それを1つずつワイルドカード展開をします。
そしてその結果を、一文字の空白をはさんで結合します。
〈リスト〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。
ここでのワイルドカードの意味は、Human68k での意味と同一です。

$(calculate 〈式〉)

〈式〉を評価した結果を展開します。
〈式〉は if 指令の〈式〉と同じです。
「指令行」の説明を参照してください。
〈式〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

●フィルタコマンド

マクロ呼び出しの( )の中に文字'｜'があれば、それ以降はフィルタコマンドとして認識されます。
フィルタコマンドは"｜..｜..｜.."の様にいくつでも連結することが可能です。
フィルタコマンドには次のものがあります。

addprefix
addsuffix
basename
dir
extract
findstring
head
sort
strip
subst
suffix
tail
tolower
toupper
uniq
word
words

これらを使う時は、小文字でなければいけません。
また、スイッチがあるフィルタコマンドにスイッチを指定する場合は、フィルタコマンドに続けて”ー〈スイッチ〉”と指定してください。
たとえばsubstコマンドにスイッチに‘i’と‘s’を指定する場合には以下のようになります。

”subst－is”

このときコマンドと‘ー’、‘ー’とスイッチの間に空白を入れてはなりません。
以下に各フィルタコマンドについて説明します。

addprefix〈接頭辞〉

文字列を空白で区切り、それぞれの単語の前に〈接頭辞〉を結合し、1文字の空白で区切って出力します。
〈接頭辞〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

addsuffix〈接頭辞〉

文字列を空白で区切り、それぞれの単語の後ろに〈接尾辞〉を結合し、1文字の空白で区切って出力します。
〈接尾辞〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

basename

文字列を空白で区切り、それぞれの単語から接尾辞を取り除いた部分を1文字の空白で区切って出力します。

dir

文字列を空白で区切り、それぞれの単語のディレクトリ部分を、1文字の空白で区切って出力します。
ディレクトリ部分は最後の‘¥’を(もしあれば)を含みます。
単語がディレクトリを含まない場合には‘.¥’を出力します。

extract〈パターン〉

文字列を空白で区切り、〈パターン〉に適合する単語をふるいにかけます。
単語間の空白は保存されません。
〈パターン〉が、マクロ呼び出しを含んでいるならば、それは事前に展開されます。

〈パターン〉は次のメタ文字が使用できます。

?	任意の1文字に適合する
[list]	list 中の1文字に適合する
[lower-upper]	lower と upper の範囲にある文字に適合する
*	(空白を含む)任意のパターンに適合する
¥	続く文字をクォートする

この〈パターン〉とは、'%'によるパターンマッチや、Human68k のワイルドカードではありません。
extract には次のスイッチがあります。

i …………大文字と小文字を区別しない
v ………… 〈パターン〉に適合する単語を削除する

スイッチは、"extract—iv" のように指定します。

v スイッチが指定されていない場合には、〈パターン〉に適合しない単語を削除します。

findstring 〈検索文字列〉
文字列から〈検索文字列〉を検索する。
もし見つかれば、結果は〈検索文字〉となり、見つからなければ空文字列となります。
〈検索文字列〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。
findstring には次のスイッチがあります。

i…………大文字と小文字を区別しない
スイッチは、"findstring—i" のように指定します。

head
文字列を空白で区切り、それぞれの単語のディレクトリ部分を、一文字の空白で区切って出力します。
ディレクトリ部分は最後の'¥'を含みません。
単語がディレクトリを含まない場合には単語をそのまま出力します。

sort

文字列を空白で区切り、語単位で文字コード順にソートします。
単語間の空白は保存されません。
sort には次のスイッチがあります。

i…………大文字と小文字を区別しない
r…………逆順にソートする

スイッチは、"sort－fr" のように指定します。

strip

文字列中の連続した空白を1個の空白に置き換え、文字列の先頭と末尾の空白を削除します。

subst 〈検索文字列〉 ＝ 〈置換文字列〉

文字列中の〈検索文字列〉に適合する部分を〈置換文字列〉で置換します。
〈検索文字列〉および〈置換文字列〉がマクロ呼び出しを含んでいるならばそれは事前に展開されます。
subst には次のスイッチがあります。

i…………大文字と小文字を区別しない
s…………語末についてのみ検索する
p…………パターン検索を行う

スイッチは、"subst－is" のように指定すます。
p スイッチが指定されたときには空白で区切られた語単位で検索します。
もし〈検索文字列〉が'%'を含んでいれば、検索された単語の'%'に適合する部分が〈置換文字列〉の'%'の部分に置き換わります。
置換後、文字列中の連続した空白は1個の空白で置き換えられます。

suffix

文字列を空白で区切り、それぞれ単語の接尾辞の部分を、1文字の空白で区切って出力します。
単語が接尾語を含まない場合には単語は消滅します。

tail

文字列を空白で区切り、それぞれの単語のファイル名部分を、1文字の空白で区切って出力します。

tolower

文字列中の大文字のアルファベットをすべて小文字に変換します。

topperr

文字列中の小文字のアルファベットをすべて大文字に変換します。

uniq

文字列を空白で区切り、同一の単語が連続していれば重複分を削除します。

単語間の空白は保存されません。

usiq には次のスイッチがあります。

f …………大文字と小文字を区別しない
d …………同一の単語が連続していない部分を削除する
u …………連続している同一の単語をすべて削除する

スイッチは、”uniq—fd”のように指定します。

words

文字列中の空白で区切られた単語の数を出力します。

word 〈パラメータ〉

〈パラメータ〉に従って文字列中の単語を取り出します。

単語間の空白は保存されません。

〈パラメータ〉がマクロ呼び出しを含んでいるならば、それは事前に展開されます。

〈パラメータ〉は次の形式で指定します。

— …………すべての単語を出力する
n ………… 〈n〉番目の単語を出力する
n— ……… 〈n〉番目から最後まで単語を出力する
—n ………最後から〈n〉番目までの単語を出力する
n—m …… 〈n〉番目から〈m〉番目までの単語を出力する

1.2　プログラム保守ユーティリティ MAKE

●メイクファイル

メイクファイルは次の五つの部分から構成されます。

- 空白行
- コメント行
- マクロ定義行
- 規則行
- 指令行

以下、各要素について説明します。

●空白行

空白のみの行は何も意味を持ちません。

●コメント(注釈)行

行の先頭に、または空白に続いて文字'#'があるとき、'#'から行末まではコメントとみなし無視されます。
ただし、行末の'¥'は、行継続として機能しますので注意してください。

●マクロ定義行

メイクファイル及び MAKE コマンドでのマクロ定義は次のように記述されます。

```
〈マクロ名〉 = 〈値〉または、〈マクロ名〉:= 〈値〉
```

［解説］

- 行の先頭に余分な空白があっても無視されますが、先頭の文字がタブだとコマンドと見なされますのでタブを書いてはいけません。
- '='および':='の前後の空白は無視されます。
- マクロ名は'='および':'を含んではなりません。
- マクロ名の大文字と小文字は区別されます。

・'='による定義では、〈値〉がそのまま定義されます。
〈値〉にマクロ呼び出しが含まれていれば、それはこのマクロが呼び出されたときに展開されます。

・':='による定義では、〈値〉に含まれるマクロ呼び出しを展開してから定義されます。このマクロが呼び出されると、単にそれを返します。
つまりそれ以上展開されません。

●**規則行**

1) 規則の記述方法

規則は次のように記述されます。

```
〈ターゲット名〉… {: | ::} [〈パターン〉:] 〈依存ファイル〉…;〈コマンド〉
〈タブ〉〈コマンド〉
        ・
        ・
        ・
```

ターゲット名：作成する目的ファイル名
パターン　　：明示的なパターン規則の時使用
依存ファイル：ターゲットが依存しているファイル名
コマンド　　：ターゲットを更新するために実際に行われるコマンド
行末で';'に続けて書く書式と、次の行にタブを続けて書く書式の２つがあります。
タブ：TABキーを押すと入力されるキーコード

規則は、ターゲットファイルとその構成要素であるソース・ファイルとの依存関係と、ターゲット作成のためのコマンド行が記述されます。
規則を記述する上での注意事項を以下に示します。

・それぞれの行の末尾の空白は削除されます。
・ターゲット名は空白で区切ります。
・ターゲットが依存しているものの名前は空白で区切ります。
・ターゲット名に続く':'には空白を続けなければなりません（行末ならば不必要です）。
ただし、'::'ならば空白は必要ありません。
・メイクファイルは行単位で解析されます。
もし末尾の文字が'¥'ならば次の行も含めて１行として扱います。

もし行末尾に単なる文字としての'¥'を書きたいならば、それは'¥¥'と書かなければなりません。
ただし、行継続は、このMAKEでは必要ありません。
物理的な行の長さに制限は無いからです。
メイクファイルを読み易くするためや、エディタで編集し易くする為に行ってください。

- 行継続が行われるとき、行末の改行と次の行の先頭に並ぶ空白は1個の空白に置き換えられます。
  したがって行継続をする場合には空白の位置で分割しなければなりません。
- パターンに続く':'が行末でない限り、空白を続けなければなりません。
- 暗黙のパターン規則には" 〈パターン〉 :"があってはなりません。
- ' 〈パターン〉 :'を含む規則は明示的なパターン規則になります。
- ターゲット名の大文字と小文字は区別されません。また、'¥'と'／'も区別しません(特殊ターゲットパターンも)。
- ／f スイッチの指定や include 指令行や MAKEFILES などでファイル名として'_'を指定すると標準入力となります。
- 最初の行の先頭に余分な空白があっても無視されます。
  ただし、先頭の文字がタブだとコマンド行として認識されますので注意してください。
- コマンドは最初の行の終わりに';'をつけて書くか、上述のように、行頭でタブに続けて書きます。
- コマンドに含まれるマクロ呼び出しは、コマンドが実行される時点で展開されます。それ以外の部分に現れるマクロ呼び出しは、この規則を読み込んだ時点ですぐに展開されます。
- ターゲットおよびターゲットが依存しているものの名前は、ワイルドカードを含んでいてもかまいません。
  ワイルドカードはこの規則を読み込んだ時点ですぐ展開されます。
  ここでのワイルドカードの意味は Human68k でのワイルドカードと同一です。
- コマンドは1行ずつ COMMAND. X に渡されます。
- ターゲットはメイクファイルの最初で定義されなければなりません。
- ターゲットが依存しているものは自動的に新たなターゲットとなります。
- コマンドが記述されていなければ暗黙のパターン規則の適用を受けます。
- コマンド行の先頭には'@'および'−'を書くことができ、以下の意味になります。

@…………コマンド行を画面に表示することを止める

指定したコマンドのみ有効

—…………MAKE は終了コードをチェックしない

終了コードにかかわらず、MAKE は実行を続ける

2) 規則には次の種類があります。

・通常の規則
・明示的なパターン規則
・暗黙のパターン規則
・暗黙の接尾辞規則

3) 通常の規則

一般的なもので、1)の文法に従います。

4) 明示的なパターン規則(static pattern rule)

パターンは必ず1個の'%'を含みます。
MAKE はターゲット名とパターンと比較し、'%'が適合するターゲット名の部分を求めます。
そしてもしターゲットが依存しているものの名前が'%'を含んでいたら、それを置き換えます。
パターンは必ずターゲット名に適合しなければなりません。

［例］

foo.o:	%. o: %. c	"%. c" は "foo.c" となる
foobar.o:	foo%. o: %. c	"%. c" は "bar.c" となる
¥bar¥foo.o:	%. o %. c	"%. c" は "¥bar.¥foo.c" となる
¥bar¥foo.o:	¥bar¥%. o: %. c	"%. c" は "foo.c" となる

'%'に適合した部分は、コマンド行では "$*" で参照することができます。
パターンがある以外は、書き方の意味は通常の規則と同じです。
明示的なパターン規則であってもコマンドが記述されていなければ、暗黙のパターン規則の適用を受けます。

5) 暗黙のパターン規則

ターゲット名は必ず1個の'%'を含みます。
次の条件がすべて満たされている時、暗黙の規則が検索されます。

・更新のためのコマンドが記述されていない
・ターゲットは phony ターゲットではない
・ターゲットは終端ターゲットではない

探索の結果、もし見つかればそれが適用されますが、見つからなければ、.DEFAULT のコマンドが(もしあれば)使用されます。

［例］ main. c、subl. c、sub2. s の３つのファイルがあったとします。
このとき、以下のような規則があった場合、

```
LIB＝CLIB. L
%. o：：%. c
        CC／Fc $<
%. o：：%. s
        AS $<
main. x：  main. o sub1. o sub2. o
        LK $^$(LIB)
```

まず最終ターゲットは、通常の規則の最初に現れたターゲットであるので、main. x が最終ターゲットとして判断されます。
次に、それが依存しているファイル main. o、sub1. o、sub2. o が次のターゲットとなります。
main. c が依存しているファイルは、暗黙のパターン規則により、main. c か main. s のどちらかになります。
MAKE は、１つ目に記述されている暗黙のパターン規則を調べ、main. c がファイルとして存在するので、それを採用します。同様に sub1. o の依存ファイルを sub1. c として判断します。
しかし、sub2. o の依存ファイルを推論する場合は１つ目のパターン規則で推論した sub2. c はファイルとして存在しません。
そこで、MAKE は１つ目の暗黙のパターン規則を採用せず、２つ目の暗黙のパターン規則から、sub2. o の依存ファイルは sub2. s であると判断します。
以上のようにして推論を行いながら、暗黙のパターン規則の部分に記述されていたコマンドを実行し、main. o、sub1. o、sub2. o を生成します。

```
        CC／Fc   main. c
        CC／Fc   sub1. c
        AS   sub2. s
```

そして、main. x の依存ファイルである、main. o、sub1. o、sub2. o が生成された次に、main. x を生成するためのコマンドを実行します。

```
        LK   main. o   sub1. o   sub2. o   CLIB. L
```

上記の例で、暗黙のパターン規則で'：：'と記述されているのは、「終端の規則」を表していて、main. c や sub2. s などが、それ以上依存関係がないことを意味しています。

これは、安全のため、高速化のために、それ以上推論を行わないように記述しています。

6)暗黙の接尾辞規則

暗黙の接尾辞規則は歴史的な互換性のために認めています。
出来るだけ、5)の方法を使うようにしてください。
このMAKEでは、メイクファイルに暗黙の接尾辞規則が記述されていたら、それを暗黙のパターン規則に変換します。

［例］
```
.SUFFIXES:  .x  .o  .s  .c
.c. o:      $(CC)  $(CFLAGS)  $<
```

以上のメイクファイルは、「暗黙のパターン規則」では、

```
%.o: %.c
        $(CC)  $(CFLAGS)  $<
```

となります。
これは、特殊ターゲット".SUFFIXES"で制御しなければ使えません。
同じ暗黙の接尾辞規則が現れ、どちらもコマンドを持っているとき、後に現れた方のコマンドが採用されます。

●指令行

指令には次のものがあります。

```
if～elif～else～endif
include
define～endef
override  define～endef
undef
vpath
error
```

これらを使うとき小文字でなければなりません。
指令行の先頭には空白があってもかまいませんが、タブで始まってはなりません。
以下、各指令行について説明します。

1)　if 〈式〉　～ elif 〈式〉　～ else　～ endif
これらのうち、elif、else はなくてもかまいません。
if＜式＞で、＜式＞が真ならば、＜式＞の次から elif、else、endif が現れるまでを、真でないならば、else から endif までまたは else 以降を実行します。
elif は、else if の意味で、「そうでないならばもし」ということになります。
また、elif はいくつあってもかまいません。
入れ子にすることが可能です。
if, elif に続く〈式〉について説明します。
〈式〉にマクロ呼び出しを含んでいるならば、それは事前に展開されます。

・数値演算子
数値演算の項は 10 進数で表された文字列でなければなりません。
結果もやはり数値を 10 進数で表した文字列になります。

＋〈式〉
－〈式〉
～〈式〉
！〈式〉
〈式 1〉＋〈式 2〉
〈式 1〉－〈式 2〉
〈式 1〉 * 〈式 2〉
〈式 1〉／〈式 2〉
〈式 1〉 % 〈式 2〉
〈式 1〉 & 〈式 2〉
〈式 1〉 ^ 〈式 2〉
〈式 1〉 | 〈式 2〉
〈式 1〉 && 〈式 2〉

〈式 1〉 || 〈式 2〉

〈式 1〉 << 〈式 2〉

〈式 1〉 > 〈式 2〉

〈式 1〉 < 〈式 2〉

〈式 1〉 > 〈式 2〉

〈式 1〉 <= 〈式 2〉

〈式 1〉 >= 〈式 2〉

これらはCコンパイラと同じ結果を返します。

・文字列比較演算子

〈式 1〉 == 〈式 2〉

〈式 1〉 != 〈式 2〉

〈式 1〉 と 〈式 2〉 を文字列として比較します。
大文字と小文字は、区別されます。
結果が真ならば'1'、さもなくば'0'となります。
〈式 1〉 と 〈式 2〉 が数値を表している場合も、あたかも数値演算のように比較されます。

〈式 1〉 =‾ 〈式 2〉

〈式 1〉 !‾ 〈式 2〉

〈式 2〉 にメタ文字('*'、'?'、'[ ]'、および'¥')を使用したパターンマッチングを行います。
結果が真ならば'1'、さもなくば'0'となります。

・ファイル検査演算子

ファイル検査演算では、〈式〉 をパス名として検査を行います。

－a 〈式〉 ………アーカイブビットが1ならば'1'、0ならば'0'となります。存在しなければ'0'となります。

－d 〈式〉 ………ディレクトリならば'1'、さもなくば'0'となります。存在しなければ'0'となります。

－e 〈式〉 ………存在すれば'1'、さもなくば'0'となります。

－f 〈式〉 ………ファイルならば'1'、さもなくば'0'となります。存在しなければ'0'となります。

−h 〈式〉 ………隠されたファイルならば'1'、さもなくば'0'となります。存在しなければ'0'となります。

−s 〈式〉 ………システムファイルならば'1'、さもなくば'0'となります。存在しなければ'0'となります。

−w 〈式〉 ………書き込みが可能ならば'1'、禁止されていたら'0'となります。存在しなければ'0'となります。

−z 〈式〉 ………大きさがが0ならば'1'、さもなくば'0'となります。存在しなければ'0'となります。

・その他

( ) が使用できます。

マクロを呼び出す場合には、以下の事項に注意してください。

例えば、

$(MESSAGE)＝HELLO

では、マクロが未定義か、空文字列が設定されているか、もしくは設定されている値が空白だけならば、

＝HELLO

となってしまいます。

この式はエラーです。

"$(MESSAGE)"＝"HELLO"

と、'か"でくくって書かなければなりません。

また、式に現れる"および'は外されてから演算が行われます。

"および'は、前に¥を付け¥"か¥'にしてください。

2)　include 〈パス名〉

ファイルをインクルードします。

もし〈パス名〉で指定されたファイルが見つからなかったときには、〈パス名〉が絶対パス名でなければ／Iスイッチで作られたパスを順次〈パス名〉の前に加えて検索します。それでも見つからなければエラーになります。

〈パス名〉にマクロ呼び出しがを含んでいるならば、それは事前に展開されます。

3) define 〈マクロ名〉 ~endef

define 指令行の直後の行から、endef 指令行の直前の行までに記述された文字列を、マクロ〈マクロ名〉にレベル3で定義します。

文字列の改行や空白はすべてそのまま定義されます。

4) override 〈マクロ名〉 = 〈値〉
   override 〈マクロ名〉 := 〈値〉

定義レベルが6となる以外は通常のマクロ定義と同一です。

5) override define 〈マクロ名〉 ~endef

定義レベルが6となる以外は、3)の define 指令行と同一です。

6) undef 〈マクロ名リスト〉

〈マクロ名リスト〉は空白で区切られたマクロ名のリストです。

記述されたマクロ名の定義をすべて削除します。

ただし、英数字または'‾'で始まらない名前の定義は削除しません。

記述されたマクロ名が定義されていない場合はそれを無視します。

7) vpath [〈パターン〉 [〈パスリスト〉]]

パターンに適合するファイルについての検索パスを指定します。

マクロ"VPATH"より選択的な検索パスを設定できます。

〈パスリスト〉は';'で区切ります。

〈パターン〉には次のメタ文字を使用できます。

?	任意の1文字に適合する
[list]	list 中の1文字に適合する
[lower-upper]	lower と upper の範囲にある文字に適合する
*	(空白を含む)任意のパターンに適合する
¥	続く文字をクォートする

ここでの〈パターン〉は'%'によるパターンマッチや、Human68k のワイルドカードではないことに注意してください(マクロフィルタ"extract"のパターンと同じです)。

もし同じ〈パターン〉の検索パスがすでに設定されていても、この指定はそれに付け加えるものではありません。

新たに置き換えられるだけです。

〈パスリスト〉が記述されていない場合には〈パターン〉の検索パスを消去

します。
〈パターン〉も記述されていない場合にはすべての検索パスを消去します。
ただし、いずれの場合にもマクロ"VPATH"による設定は残ります。
〈パターン〉や〈パスリスト〉にマクロ呼び出しを含んでいるならば、事前に展開されます。

8)　error［〈メッセージ〉］
〈メッセージ〉を表示して処理を中止します。
〈メッセージ〉にマクロ呼び出しを含んでいるならば事前に展開されます。

●特殊ターゲット

特殊ターゲットには次のものがあります。

.SUFFIXES ……暗黙の接尾辞規則で、複数の接尾辞のリストを書くと、ターゲットファイルの接尾辞をもとに、リストの順番で接尾辞を持つファイルをさがし、見つかったものよりファイル名と作成規則を推論します。
他のCコンパイラの暗黙の規則を使用したメイクファイルは、これを定義しておかないと使用できないことがあります。

［例］　.SUFFIXES：　.o　.c

.DEFAULT ……メイクファイル内に、ターゲットファイルを作るためのコマンドが指示されていない時、または、ファイルを作るための規則がない時、".DEFAULT"以後に指示されたコマンドがあればそれが使用されます。

.IGNORE ………MAKEコマンドの／iスイッチを指定した時と同じです。

.PHONY ………このターゲットが依存しているものは、phonyターゲットにします。
phonyターゲットは実際のファイルとは無関係に取り扱われます。

.PRECIOUS ……このターゲットに依存するファイルを暗黙的に作成した場合、自動的に削除するのを禁止します。

.SILENT ………MAKEコマンドの／sスイッチを指定した時と同じです。

すなわち、

［例］ PHONY: ALL
All : FOO. X BAR. X

もしファイル"ALL"が存在していてもそれは無関係とし、無視します。
したがってターゲット"ALL"は必ず更新されます。
phonyターゲットはpreciousとして扱われます。
また、／tスイッチでも日付を更新しません。

また、phonyターゲットには暗黙の規則は適用されません。

.TERMINAL …このMAKE独自のもので、このターゲットに依存しているものは、終端ターゲットになります。
終端ターゲットは暗黙の規則は適用されません。
この機能は安全のため、また、高速化のために用意されています。

● **MAKEコマンドの記述**

MAKE ［〈スイッチ…〉］ ［〈マクロ定義〉］ ［〈最終ターゲット名〉］ ……

［例］ MAKE /s /n VPATH=SOURCE TEST. X ⏎

- /s /n ― スイッチ
- VPATH=SOURCE ― マクロ定義
- TEST. X ― 最終ターゲット名

1.2　プログラム保守ユーティリティ MAKE

［解説］

MAKE コマンドはパラメータを空白で区切り、それぞれをスイッチ・マクロ定義・ターゲット名のいずれかに分類します。
空白を含む語は"で囲んで下さい。
MAKE は'／'または'－'で始まる語をスイッチとしてと処理し、それ以外のパラメータのうちメイクファイルのマクロ定義と同じ形式のものをマクロ定義として処理します。
そして最後に残ったパラメータは最終ターゲット名であると処理します。
スイッチは左から順に処理されます。
コマンドに／f スイッチが一つも指定されていない場合、"makefile"をデフォルトファイルとして読み込みます。

コマンドに最終ターゲット名が一つも指定されていない場合、メイクファイルに現れた最初の暗黙の規則でない規則の、最初のターゲットを最終ターゲットとします。
ただし、'.'で始まり、しかも先頭が".¥"や"..¥"でないターゲットは除外されます。
また、include 指令行や MAKEFILES で読み込まれたファイルでは最終ターゲットは決定されません。

● MAKE コマンドスイッチ・リファレンス

MAKE コマンド行に記述できる全スイッチを以下に示します。スイッチでは、大／小文字で区別がありますので注意してください。

／a ……………… タイムスタンプを無視し、最終ターゲット及び全ての従属しているもののうち依存しているものを持つものを必ず更新します。
このスイッチは実際に全ての更新をやり直す為に使うべきではありません。
なぜならエラーなどで更新処理を途中で中断するたびに／a スイッチを付けて最初から全部をやり直さなければならず、非常に効率が悪くなってしまいます。
この MAKE が何をするかを知ったり、出力をリダイレクトしてバッチファイルやドキュメントを作成したりする場合のために／n スイッチと組み合わせて使用してください。

／c path…………カレントディレクトリを続く引数で指定されたディレクトリに変更します。

／d ………………デバック情報を表示します。
これは、MAKE そのもののデバック情報を得る為のものではなく、ユーザーが作成したメイクファイルに誤りが無いかどうかを知る為のオプションです。
メイクファイル、コマンド行、環境変数、ディレクトリ内容などが、どのような状態で、どのように解釈され、どのようなことが起こったかを知るためのものです。
MAKE の動作が期待した通りでなかった時にこのスイッチを使用してメイクファイルを修正する情報として活用してください。

／D ………………ターゲットについての考査を終えるたびに、ファイルに関するデータベースを標準出力に出力します。

／e ………………環境変数がメイクファイルでのマクロ定義を覆うようにします。
このスイッチを多用して、もし、異なった作業環境で MAKE を使用すると、期待した動作を行わない場合があります。
なるべくこのスイッチは使用せず、どのような作業環境でも使用できるメイクファイルを記述してください。

／f file …………続く引数で指定されたファイルをメイクファイルとして読み込みます。
file に'－'を指定すると標準入力からの読み込みになります。

／h ……………… MAKE のコマンド行の書式と全スイッチの説明を表示し、他には何もせずに終了します。

／i ………………更新のために実行したコマンドで発生したエラーを全て無視します。
このスイッチは使うべきではありませんが、歴史的な互換性のために存在します。
また、正常に終了しても、コマンドの終了コードが正常終

了にならないコマンドを使用する時は、MAKEが処理を中断しないようにするために、このスイッチを指定してもかまいません(純正コマンドを使用している場合にはこのようなことは在り得ません)。

／I path…………続く引数で指されたディレクトリを、メイクファイル中のinclude指令で検索するディレクトリのリストに追加します。

／k ………………エラーのために更新を中断しても、影響のない他のターゲットの考査を続行します。

／n ………………更新のためのコマンドを実行しません。
ターゲットを更新する心要があれば更新のためのコマンド行を標準出力へ出力するだけです。
ただしコマンド行が"$(MAKE)"または"${MAKE}"を含んでいる場合には必ず実行します。なお、コマンド行の表示は、'@'付きのコマンド行であっても、また、MAKEが／sスイッチ付きで起動されていても表示されます。

／N………………考査を行いません。
このスイッチは必ず／pスイッチと同時に使用してください。

／o name ………続く引数で指定されたターゲットを考査しません。
MAKEはこのスイッチで指定されたターゲットを「最も遠い過去に既に更新されたもの」とみなします。
したがって指定したターゲットは決して更新されず、そのターゲットに依存するターゲットも影響を及ぼされません。

／p ………………メイクファイル、環境変数、コマンド行などから構築されたデータベースを標準出力に出力します。
単にデータベースを出力したいだけならば、同時に／Nスイッチを指定してください。そうすることによって、データベースを表示する以外には何もせずに終了します。

／q ………………考査の結果、更新するべきターゲットがあれば1を、すべてのターゲットがすでに更新されていたならば0を、終了コードとして親プロセスに返します。
更新は行いません。

／R name ………続く引数で指定されたターゲットを必ず更新します。
MAKEはこのスイッチで指定されたターゲットを存在しないものとみなします。
したがって指定したファイルは必ず更新します(ただし、phonyターゲットとは意味が異なります)。

／s ………………更新のためのコマンドを起動するとき、コマンド行を表示しません。
ただし、／nスイッチが指定されている場合には、このスイッチは無視されます。

／S ………………以前に指定された／kスイッチを取り消します。

／t ………………更新のためのコマンドを実行するかわりに、単にファイルの作成日付のみを更新します。

／v ……………… MAKEのバージョンについての情報を表示します。

／w ……………… MAKEの処理の前後に、カレントディレクトリを表示します。

／W name………続く引数で指定されたターゲットを新しいものとみなします。
したがって指定したファイルは更新されませんが、このターゲットに依存するターゲットは（／oスイッチで指定したターゲットでなければ）必ず更新されることになります。
このスイッチと／nスイッチとを組み合わせると、あるファイルを更新すると「何が起こるか」を知ることができます。
もちろん、あるターゲットを更新直後であるとみなして更新処理を行いたい場合にも使用できます。

●マクロ定義

MAKE コマンドのマクロ定義の書式はメイクファイルの書式と同じです。
詳しくは「●マクロ定義行　P43」を参照してください。
マクロ定義全体については「1.2.3 メイクファイルのサンプル」を参照してください。

第2章

DOS コール

DOS コールの種類

DOS コールの詳細

X68000では、システムを管理したり主な基本動作を行う標準的なサブルーチンをファンクションコールと呼び、その仕様を公開しています。
ファンクションコールはユーザーの作成したプログラムからコールすることができます。
ユーザーがX68000上（Human68k上で）動作するソフトウェアを開発する際に、これらのファンクションコールを使用すれば、より汎用性の高いプログラムを、より簡単に作成することができます。
ファンクションコールは、主要な基本動作を実現するためのサブルーチン集ともいえるものです。
X68000のファンクションコールには、大きく分けて2種類あります。
ひとつは、ROM内に用意されている（拡張BIOSではメモリ上）IOCSコールです。
もうひとつは、OSであるHuman68kに用意されているDOSコールです。
本書では、この第2章でDOSコールを、次の第3章でIOCSコールを解説します。

2.1 DOSコールの種類

本章で解説する Human68k の DOS コールは、次のように分類できます。

・キャラクタデバイスからのI/O
・メモリの管理
・プロセスの管理
・ファイルやディレクトリの管理
・その他の DOS コール

2.1.1 キャラクタデバイスからのI/O

キャラクタデバイスとは、一度に1バイトずつ読み書きするデバイス（装置）のことです。
これに対して、ディスクのように一度にまとまった量のデータを読み書きするデバイスのことを、ブロックデバイスと呼びます。
Human68k における標準的なキャラクタデバイスは、コンソール、プリンタ、シリアルポートなどです。
クロック（内部時計）をキャラクタ型デバイスに含むこともありますが、本書では「その他の DOS コール」に含めています。
DOS コールを使用すれば、これらのデバイスを同じように取り扱いデータを入出力することができます。
また、デバイスによっては、入力／出力の際にリダイレクトをすることもできます。
次の表は、標準デバイスからのI/Oを取り扱う DOS コールの一覧です。
互いによく似た機能を持ったコールもありますが、コントロールキャラクタを認識するかどうかなどで区別できます。
詳細は各コールの解説をご覧ください。

コール	名 前 と 機 能		ページ
$FF01	getchar	キーボードの入力を待つ（エコーつき）	81
$FF02	putchar	文字コードにより文字を表示	82
$FF03	cominp	RS-232C回線から1バイト入力	83
$FF04	comout	RS-232C回線から1バイト出力	84
$FF05	prnout	プリンタに1文字出力	85
$FF06	inpout	キーの入出力	86
$FF07	inkey	キーボードから1文字入力（ブレークチェックなし）	87
$FF08	getc	キーボードから1文字入力（ブレークチェックあり）	88
$FF09	print	文字列を表示	89
$FF0A	gets	文字列を入力	90
$FF0B	keysns	キーの入力状態をチェック	91
$FF0C	kflush	バッファフラッシュつきキーボード入力	92
$FF10	consns	画面出力ができるかどうかをチェック	100
$FF11	prnsns	プリンタ出力ができるかどうかをチェック	101
$FF12	cinsns	RS-232C回線から入力の可・不可のチェック	102
$FF13	coutsns	RS-232C回線への出力の可・不可のチェック	103
$FF18	hendsp	漢字変換行をコントロール	107
$FF1A	getss	文字列を入力	111
$FF22	knjctrl	かな漢字変換ファンクションコール呼び出し	121
$FF23	conctrl	コンソールに直接出力	122
$FF24	keyctrl	コンソールに直接入力	128
$FF44	ioctrl	デバイスドライバのIOCTLによる入出力	165

2.1.2 メモリの管理

Human68kでは、メモリ管理ポインタと呼ばれるポインタでメモリを管理しています。

メモリ管理についての詳細は、本書第7章「7. 2　メモリ管理ポインタ」を

ご覧ください。
次の表は、メモリ管理に関するDOSコールの一覧です。

コール	名前と機能	ページ
$FF48	malloc メモリの確保	172
$FF49	mfree メモリブロックを開放	173
$FF4A	setblock メモリブロックを変更	174
$FF58	malloc2 メモリの確保	196

2.1.3 プロセスの管理

Human68kでは、プロセスがメモリにロードされる際、上記のメモリ管理ポインタとともにプロセス管理ポインタを用いて環境のアドレス、さまざまな場合に応じた戻りアドレス、コマンドラインのアドレスなどを記憶しています。
次表のようなDOSコールを使用すれば、ユーザーが作成するアプリケーションプログラムにおいても、プロセス管理ポインタを特に意識することなくプロセスをハンドリングできます。
プロセス管理についての詳細は、本書第7章「7.3 プロセス管理ポインタ」をご覧ください。
次の表は、プロセスの管理に関するDOSコールの一覧です。

コール	名前と機能	ページ
$FF00	exit プログラムを終了	80
$FF26	pspset バッファに終了情報を書き込む	132
$FF31	keeppr 常駐終了	144
$FF4B	exec プログラムのロード／実行	175
$FF4C	exit2 終了コードを指定して終了	181
$FF4D	wait プロセスの終了コードを得る	182
$FF50	setpdb 管理プロセスを移す	186
$FF51	getpdb 現在のプロセス情報を得る	187
$FF52	setenv 環境変数を設定	188
$FF53	getenv 環境変数の内容を得る	189

2.1.4 ファイルやディレクトリの管理

Human68kでファイルをアクセスする際には、ファイルハンドルと呼ばれる番号を使用します。
ファイルハンドルの番号は1ワード(2バイト)の変数として扱われ、その考え方はちょうどBASICのファイル番号(fopen関数を参照)に似ています。
ファイルハンドルは、create($FF3C)、またはopen($FF3D)でファイルを作成(オープン)する際、そのファイルの属性を渡し、そのファイルが正常にオープンできたときファイルに個別に与えられます。
それ以後、オープンしたファイルはすべてそのファイルハンドルを用いてアクセスされます。
Human68kではコンソール、プリンタなどのデバイスもファイルと同様に扱われるため、これらのデバイスが予約ファイル名でオープンされた場合にも、同じようにファイルハンドルが与えられます。
次の5つのファイルハンドルは標準でオープンされている(システム側で予約している)ため、これらを使用する際にはあらためてオープンする必要がありません。
なお、標準入出力、リダイレクトなどに関しては本体添付の「Human68kユーザーズマニュアル」をご覧ください。

ハンドル	名前と標準デバイス名
0	標準入力(通常はCON、リダイレクト可)
1	標準出力(　　〃　　)
2	標準エラー出力(つねにCON)
3	標準補助装置(AUX)
4	標準プリンタ(PRN)

Human68kでは、普通ファイルを作成(オープン)する際には、まだ利用していない(あるいはすでに解放された)ファイルハンドルのうちから最小のものを、ファイルハンドルに割りあてます。
ファイルハンドルの値は0からCONFIG.SYSの"FILES="で決めた値まで(最大93個まで)ですが、標準でオープンされているハンドルが5つあるため、ユーザープログラムで得られるファイルハンドルは、"FILES="

で決めた値−5となります。

ファイルはデータの行列として扱われますので、読み書きするファイルハンドル、バイト数、読み込み／書き込み先のポインタを指定するだけで、ファイルを読み書きできます。

次の表は、ファイルを管理するDOSコールの一覧です。

コール	名前と機能	ページ
$FF1B	fgetc　ファイルハンドル1バイト入力	112
$FF1C	fgets　ファイルハンドルから文字列を入力	113
$FF1D	fputc　ファイルハンドルに1バイト出力	114
$FF1E	fputs　ファイルハンドルに文字列を出力	115
$FF1F	allclose ファイルハンドルをクローズ	116
$FF29	namests ファイル情報をバッファに展開	135
$FF2F	dup0　ファイルハンドルを強制複写	142
$FF37	nameck ファイル情報の展開	151
$FF3C	create　ファイルを作成	156
$FF3D	open　ファイルをオープン	157
$FF3E	close　ファイルハンドルをクローズ	159
$FF3F	read　ファイルから読み込む	160
$FF40	write　ファイルへ書き出す	161
$FF41	delete　ファイルを削除	162
$FF42	seek　ファイルポインタの移動	163
$FF45	dup　ファイルハンドルを複写	169
$FF46	dup2　ファイルハンドルを強制複写	170
$FFF3	diskred ブロックデバイスから直接読み込む	208
$FFF4	diskwrt ブロックデバイスに直接書き込む	210

Human68kでは、ルートディレクトリに作成できるディレクトリエントリ数（ファイルとサブディレクトリの総称）の上限が192までと決まっています（2HDフロッピーディスクの場合)。

したがって、ルートディレクトリに入りきらないほど多くのファイル（ディレクトリ）を必要とする場合には、サブディレクトリを作成しなければなりません。

そこで、DOSコールにもディレクトリを管理するためのエントリがあります。

サブディレクトリにおいては、作成できるディレクトリエントリの数やディレクトリ構造の深さについては制限がないことになっています。
しかし、実際にディスクに記録できる記憶容量には限界があるので、物理的な限度は存在します。
次の表は、ディレクトリ管理に関するDOSコールの一覧です。

コール	名 前 と 機 能		ページ
$FF39	mkdir	ディレクトリを作成	153
$FF3A	rmdir	ディレクトリを削除	154
$FF3B	chdir	カレントディレクトリを変更	155
$FF43	chmod	ファイルの属性を変更	164
$FF47	curdir	カレントパスを得る	171
$FF4E	files	ファイル情報を検索（最初のファイル）	183
$FF4F	nfiles	ファイル情報を検索	185
$FF56	rename	ファイル名を変更（移動）	194
$FF57	filedate	ファイルの日付／時間の読み出し／設定	195
$FF5A	maketmp	テンポラリファイルの作成	197
$FF5B	newfile	ファイル名の作成	199

2.1.5 バックグラウンドの処理

Human68k ver. 2.0では、複数のプログラムを並列に実行できる「バックグラウンド処理」がサポートされています。
この機能により、指定時間にADPCMやTVのコントロールをしたり、プリントアウトしながら別の仕事を行ったりすることができます。
「バックグラウンド処理」の使い方の詳細は第8章に説明されています。
ここでは、バックグラウンド処理に関するDOSコールの一覧を示します。

コール	名 前 と 機 能		ページ
$FF7D	s_malloc	メインのメモリブロックの確保	205
$FF7E	s_mfree	メインのメモリブロックの解放	206
$FF7F	s_process	サブのメモリ管理の設定	207
$FFF8	open_pr	バックグラウンドタスクの登録	216

$FFF9	kill_pr　自分自身のプロセスの削除	219
$FFFA	get_pr　スレッドの管理情報の獲得	220
$FFFB	suspend_prスレッドをスリープ状態にする	222
$FFFC	sleep_pr　スループ状態に入る	223
$FFFD	send_pr　スレッドにコマンドやデータを送る	225
$FFFE	time_pr　現在のタイマのカウンタ値を返す	228
$FFFF	change_pr バックグラウンドタスクの切替	229

2.1.6 その他のDOSコール

その他のDOSコールには、ドライブの管理、スーパーバイザモード／ユーザーモードの切り替え、クロックなどに関するものを集めました。
次の表はその一覧です。

コール	名前と機能	ページ
$FF0D	fflush　ディスクをリセット	95
$FF0E	chgdrv　カレントドライブを指定	96
$FF0F	drvctrl　ドライブの状態をチェック／設定	97
$FF17	fatchk　ドライブのセクタが連続しているかどうかをチェック	104
$FF19	curdrv　カレントドライブを返す	110
$FF20	super　スーパーバイザモード／ユーザーモードの切り替え	117
$FF21	fnckey　再定義可能なキーの読み出し／設定	119
$FF25	intvcs　例外処理などのベクタにアドレスを設定	131
$FF27	gettim2 現在の時刻を得る	133
$FF28	settim2 時刻を設定	134
$FF2A	getdate 日付を得る	137
$FF2B	setdate 日付を設定	138
$FF2C	gettime 時刻を得る	139
$FF2D	settime 時刻を設定	140
$FF2E	verify　ベリファイフラグを設定	141
$FF30	vernum　OSのバージョン番号を得る	143
$FF32	getdpb　指定装置番号の各種情報を得る	145

$FF33	breakck ブレークチェックを設定	147
$FF34	drvxchg ドライブを入れ替える	148
$FF35	intvcg　例外処理などのベクタ値を得る	149
$FF36	dskfre　ディスクの残り容量を得る	150
$FF54	verifyg　ベリファイフラグの設定状況を得る	190
$FF55	common コモン領域の制御	191
$FF5C	lock　　ファイルのロック	201
$FFF5	indosflg OS ワークの参照	212
$FFF6	super_jsr スーパーバイザ領域のプログラムのコール	213
$FFF7	bus_err バスエラーのチェック	214

2.1.7 DOS コールエラーコード一覧

DOS コール実行時にエラーが発生した場合、リターンコードの最上位ビットが 1 にセットされます（負の数）。
そして、他のビットがエラーコードを表します。
次の表は、エラーコードの一覧です（負数のままにしてあります）。

コード	意　　味
−1	無効なファンクションコードを実行しました。
−2	指定したファイル名は見つかりません。
−3	指定したディレクトリは見つかりません。
−4	オープンしているファイルが多すぎます。
−5	ディレクトリやボリュームラベルはアクセスできません。
−6	指定したハンドルはオープンされていません。
−7	メモリ管理領域がこわされました。
−8	実行するのに必要なメモリがありません。
−9	無効なメモリ管理ポインタを指定しました。
−10	不正な環境を指定しました。
−11	実行ファイルのフォーマットが異常です。
−12	オープンのアクセスモードが異常です。
−13	ファイル名の指定に誤りがあります。
−14	無効なパラメータでコールしました。
−15	ドライブ指定に誤りがあります。

−16	カレントディレクトリは削除できません。
−17	ioctrlできないデバイスです。
−18	これ以上ファイルは見つかりません。
−19	このファイルは書き込みができません。
−20	このディレクトリはすでに登録されています。
−21	ファイルがあるので削除できません。
−22	ファイルがあるのでリネームできません。
−23	ディスクがいっぱいでファイルが作れません。
−24	ディレクトリがいっぱいでファイルが作れません。
−25	指定の位置にはシークできません。
−26	スーパーバイザ状態で、再びスーパーバイザモードの指定を行いました。
−27	同じスレッド名が存在します。
−28	プロセス間通信のバッファが書き込み禁止状態です。
−29	これ以上バックグラウンドプロセスを起動できません。
−32	ロック領域が足りません。
−33	ロックされていてアクセスできません。
−34	指定のドライブはハンドラがopenされています。
−80	ファイルが存在しています。

DOS コール一覧(名前順)

コール	名　前	ページ
$FF1F	allclose ( )	116
$FF5F	assign (MD, BUFF1 [,BUFF2] [,MODE])	202
$FF33	breakck (FLG)	147
$FFF7	bus_err (S_ADR, D_ADR, MD)	214
$FFFF	change_pr ( )	229
$FF3B	chdir (NAMEPTR)	155
$FF0E	chgdrv (DRIVE)	96
$FF43	chmod (NAMEPTR, ATR)	164
$FF12	cinsns ( )	102
$FF3E	close (FILENO)	159
$FF03	cominp ( )	83
$FF55	common (MD, NAME [,POS] [,BUFFER] [,ID_PSP] [,LEN])	191
$FF04	comout (CODE)	84
$FF23	conctrl (MD, [CODE] [,MESPTR] [,ATR] [,X ,Y] [,N] [,MOD] [,YS ,YL])	122
$FF10	consns ( )	100
$FF13	coutsns ( )	103
$FF3C	create (NAMEPTR, ATR)	156
$FF47	curdir (DRIVE, PATHBF)	171
$FF19	curdrv ( )	110
$FF41	delete (NAMEPTR)	162
$FFF3	diskred (ADR [+$80000000], DRIVE, SECT, SECTLEN)	208
$FFF4	diskwrt (ADR [+$80000000], DRIVE, SECT, SECTLEN)	210
$FF0F	drvctrl (MODE [,P1,…,Pn])	97
$FF34	drvxchg (OLD, NEW)	148
$FF36	dskfre (DRIVE, BUFFER)	150
$FF45	dup (FILENO)	169
$FF2F	dup0 (FILENO, NEWNO)	142
$FF46	dup2 (FILENO, NEWNO)	170

コール	名　前	ページ
$FF4B	exec (MD, FIL [,TYPE] [,P1, P2])	175
$FF00	exit ( )	80
$FF4C	exit2 (CODE)	181
$FF17	fatchk (FILE, BUFFER [+ $80000000,LEN])	104
$FF0D	fflush ( )	95
$FF1B	fgetc (FILENO)	112
$FF1C	fgets (INPPTR, FILENO)	113
$FF57	filedate (FILENO, DATETIME)	195
$FF4E	files (FILBUF, NAMEPTR, ATR)	183
$FF21	fnckey (MODE, BUFFER)	119
$FF1D	fputc (CODE, FILENO)	114
$FF1E	fputs (MESPTR, FILENO)	115
$FF08	getc ( )	88
$FF01	getchar ( )	81
$FF2A	getdate ( )	137
$FF32	getdpb (DRIVE, DPBPTR)	145
$FF53	getenv (GETNAME, ENV, GETBUF)	189
$FF51	getpdb ( )	187
$FFFA	get_pr (ID, BUFF)	220
$FF0A	gets (INPPTR)	90
$FF1A	getss (INPPTR)	111
$FF27	gettim2 ( )	133
$FF2C	gettime ( )	139
$FF18	hendsp (MD [, POS] [, MESPTR])	107
$FFF5	indosflg ( )	212
$FF07	inkey ( )	87
$FF06	inpout (CODE)	86
$FF35	intvcg (INTNO)	149
$FF25	intvcs (INTNO, JOBADR)	131
$FF44	ioctrl (MD [,FILENO] [,DT] [,DRIVE] [,PTR ,LEN])	165
$FF31	keeppr (PRGLEN, CODE)	144
$FF24	keyctrl (MD [,GROUP] [,INSMODE])	128

2.1　DOSコールの種類

コール	名　　前	ページ
$FF0B	keysns ( )	91
$FF0C	kflush (MODE, INPPTR)	92
$FFF9	kill_pr ( )	219
$FF22	knjctrl (MD, ??)	122
$FF5C	lock (MD, FILENO, OFFSET, LEN)	201
$FF5A	maketmp (NAMEPTR, ATR)	197
$FF48	malloc (LEN)	172
$FF58	malloc2 (MD, LEN)	196
$FF49	mfree (MEMPTR)	173
$FF39	mkdir (NAMEPTR)	153
$FF37	nameck (FILE, BUFFER)	151
$FF29	namests (FILE, BUFFER)	135
$FF5B	newfile (NAMEPTR, ATR)	199
$FF4F	nfiles (FILBUF)	185
$FF3D	open (NAMEPTR, MODE)	157
$FFF8	open_pr (NAME, COUNT, INIT_USP, INIT_SSP, INIT_SR, INIT_PC, BUFF, SLEEP_TIME)	216
$FF09	print (MESPTR)	89
$FF05	prnout (CODE)	85
$FF11	prnsns ( )	101
$FF26	pspset (PSPADR)	132
$FF02	putchar (CODE)	82
$FF3F	read (FILENO, DATAPTR, SIZE)	160
$FF56	rename (OLD, NEW)	194
$FF3A	rmdir (NAMEPTR)	154
$FF42	seek (FILENO, OFFSET, MODE)	163
$FFFD	send_pr (MY_ID, YOUR_ID, CMDNO, U_BUFF, U_LEN)	225
$FF4A	setblock (MEMPTR, NEWLEN)	174
$FF2B	setdate (DATE)	138
$FF52	setenv (SETNAME, ENV, SETLINE)	188
$FF50	setpdb (PDBADR)	186

コール	名　　前	ページ
$FF28	settim2 (TIME)	134
$FF2D	settime (TIME)	140
$FFFC	sleep_pr (TIME)	223
$FF7D	s_malloc (MD, LEN)	205
$FF7E	s_mfree (MEMPTR)	206
$FF7F	s_process (ID, START, LENGTH, I_LEN)	207
$FF20	super (STACK)	117
$FFF6	super_jsr (JMPADR)	213
$FFFB	suspend_pr (ID)	222
$FFFE	time_pr ( )	225
$FF2E	verify (FLG)	141
$FF54	verifyg ( )	190
$FF30	vernum ( )	143
$FF4D	wait ( )	182
$FF40	write (FILENO, DATAPTR, SIZE)	161

DOS コール一覧(コード順)

コール	名 前	ページ
$FF00	exit ( )	80
$FF01	getchar ( )	81
$FF02	putchar (CODE)	82
$FF03	cominp ( )	83
$FF04	comout (CODE)	84
$FF05	prnout (CODE)	85
$FF06	inpout (CODE)	86
$FF07	inkey ( )	87
$FF08	getc ( )	88
$FF09	print (MESPTR)	89
$FF0A	gets (INPPTR)	90
$FF0B	keysns ( )	91
$FF0C	kflush (MODE, INPPTR)	92
$FF0D	fflush ( )	95
$FF0E	chgdrv (DRIVE)	96
$FF0F	drvctrl (MODE [,P1,…,Pn])	97
$FF10	consns ( )	100
$FF11	prnsns ( )	101
$FF12	cinsns ( )	102
$FF13	coutsns ( )	103
$FF17	fatchk (FILE, BUFFER [+ $80000000, LEN])	104
$FF18	hendsp (MD [, POS] [, MESPTR])	107
$FF19	curdrv ( )	110
$FF1A	getss (INPPTR)	111
$FF1B	fgetc (FILENO)	112
$FF1C	fgets (INPPTR, FILENO)	113
$FF1D	fputc (CODE, FILENO)	114
$FF1E	fputs (MESPTR, FILENO)	115
$FF1F	allclose ( )	116
$FF20	super (STACK)	117
$FF21	fnckey (MODE, BUFFER)	119
$FF22	knjctrl (MD, ??)	121

コール	名 前	ページ
$FF23	conctrl (MD, [,CODE] [,MESPTR] [,ATR] [,X ,Y] [,N] [,MOD] [,YS ,YL])	122
$FF24	keyctrl (MD, [,GROUP] [,INSMODE])	128
$FF25	intvcs (INTNO, JOBADR)	131
$FF26	pspset (PSPADR)	132
$FF27	gettim2 ( )	133
$FF28	settim2 (TIME)	134
$FF29	namests (FILE, BUFFER)	135
$FF2A	getdate ( )	137
$FF2B	setdate (DATE)	138
$FF2C	gettime ( )	139
$FF2D	settime (TIME)	140
$FF2E	verify (FLG)	141
$FF2F	dup0 (FILENO, NEWNO)	142
$FF30	vernum ( )	143
$FF31	keeppr (PRGLEN, CODE)	144
$FF32	getdpb (DRIVE, DPBPTR)	145
$FF33	breakck (FLG)	147
$FF34	drvxchg (OLD, NEW)	148
$FF35	intvcg (INTNO)	149
$FF36	dskfre (DRIVE, BUFFER)	150
$FF37	nameck (FILE, BUFFER)	151
$FF39	mkdir (NAMEPTR)	153
$FF3A	rmdir (NAMEPTR)	154
$FF3B	chdir (NAMEPTR)	155
$FF3C	create (NAMEPTR, ATR)	156
$FF3D	open (NAMEPTR, MODE)	157
$FF3E	close (FILENO)	159
$FF3F	read (FILENO, DATAPTR, SIZE)	160
$FF40	write (FILENO, DATAPTR, SIZE)	161
$FF41	delete (NAMEPTR)	162
$FF42	seek (FILENO, OFFSET, MODE)	163
$FF43	chmod (NAMEPTR, ATR)	164

2.1　DOSコールの種類

コール	名　前	ページ
$FF44	ioctrl (MD [,FILENO] [,DT] [,DRIVE] [,PTR ,LEN])	165
$FF45	dup (FILENO)	169
$FF46	dup2 (FILENO, NEWNO)	170
$FF47	curdir (DRIVE, PATHBF)	171
$FF48	malloc (LEN)	172
$FF49	mfree (MEMPTR)	173
$FF4A	setblock (MEMPTR, NEW-LEN)	174
$FF4B	exec (MD, FIL [,TYPE] [,P1 ,P2])	175
$FF4C	exit2 (CODE)	181
$FF4D	wait ( )	182
$FF4E	files (FILBUF, NAMEPTR, ATR)	183
$FF4F	nfiles (FILBUF)	185
$FF50	setpdb (PDBADR)	186
$FF51	getpdb ( )	187
$FF52	setenv (SETNAME, ENV, SETLINE)	188
$FF53	getenv (GETNAME, ENV, GETBUF)	189
$FF54	verifyg ( )	190
$FF55	common (MD, NAME [,POS] [,BUFFER] [,ID_PSP] [,LEN])	191
$FF56	rename (OLD, NEW)	194
$FF57	filedate (FILENO, DATETIME)	195
$FF58	malloc2 (MD, LEN)	196
$FF5A	maketmp (NAMEPTR, ATR)	197
$FF5B	newfile (NAMEPTR, ATR)	199
$FF5C	lock (MD, FILENO, OFFSET, LEN)	201
$FF5F	assign (MD, BUFF1 [,BUFF2] [,MODE])	202
$FF7D	s_malloc (MD, LEN)	205
$FF7E	s_mfree (MEMPTR)	206
$FF7F	s_process (ID, START, LENGTH, I_LEN)	207
$FFF3	diskred (ADR [+$80000000] DRIVE, SECT, SECTLEN)	208
$FFF4	diskwrt (ADR [+$80000000] DRIVE, SECT, SECTLEN)	210
$FFF5	indosflg ( )	212
$FFF6	super_jsr (JMPADR)	213
$FFF7	bus_err (S_ADR, D_ADR, MD)	214
$FFF8	open_pr (NAME, COUNT, INIT_USP, INIT_SSP, INIT_SR, INIT_PC, BUFFER, SLEEP_TIME)	216
$FFF9	kill_pr ( )	219
$FFFA	get_pr (ID, BUFF)	220
$FFFB	suspend_pr (ID)	222
$FFFC	sleep_pr (TIME)	223
$FFFD	send_pr (MY_ID, YOUR_ID, CMDNO, U_BUFF, U_LEN)	225
$FFFE	time_pr ( )	228
$FFFF	change_pr ( )	229

2.2 DOSコールの詳細

ここでは、DOS コールの呼び出し方法、エラー処理について説明した後、DOS コールについて個々に解説します。

2.2.1 DOS コールの使い方

DOS コールは、さまざまな処理を行う Human68k のサブルーチンを呼び出します。
DOS コールを呼び出す標準的な手続きは以下の通りです。

① DOS コールの際に必要なパラメータがあれば、スタックポインタ（A7 レジスタ）にプッシュします。
②$FF??を実行します。
この??の部分が実際のコール番号になります。

MPU68000 では、

```
$FF
```

は未実装命令（未定義命令）なので、$FF に出会うとトラップが発生し、この場合は例外処理ベクタ番号 11 に書かれたアドレスから例外処理を始めます。
X68000 では、このベクタが指すアドレスに、DOS コールを処理するルーチンが置いてあります。
このルーチンは、$FF に続く 1 バイトを見て、コール番号を知り、必要に応じてスタック（A7 レジスタ）に置かれたパラメータを使い、DOS コールを実行するのです。
$FF??は、アセンブラの

```
DC. W   _(コール名)
```

のような擬似命令で書き込んでおきます。
DOS コールは、呼ばれたときの内部処理でレジスタを使いますが、なるべ

くスタックでレジスタの値を保存します。
ですから、DOSコールを終えて戻ってきたときにもD0以外のレジスタはすべて保存されています。
D0レジスタには、必要に応じてリターン値やエラーコードが入っています。
パラメータを用いたDOSコールを使用する際には、$FF??の次にスタックポインタ（A7レジスタ）の加算命令を置き、スタックの位置を呼び出す前の位置に戻しておく必要があります。

2.2.2 エラー処理

DOSコールの処理を実行中にエラーが発生すると、リターン値が入るべきD0レジスタの値がマイナスになって戻ってきます（最上位ビットが1）。
このような場合には、最上位ビットを除くビットの値がエラーコードと見なされます。
DOSコールのエラーコードの詳細については、本書2. 1. 7を参照してください。

2.2.3 DOSコールの解説方法

2. 2. 4以降では、次のような形式で個々のDOSコールについて解説します。

●名前（引数）　　　DOSコール番号：$FF??

機　能	機能の解説
引　数	引数の解説
コール	コール方法
リターン	リターン値
解　説	機能の詳細な解説

・名前（引数）

説明するDOSコールの名称です。
機能を表しています。
引数（パラメータ）があれば名前の直後にカッコ書きされます。

・DOSコール番号

DOSコールの機能番号です。

$FF??の形で書かれます。

・**機能**

機能の概要を示します。

・**引数**

もしあれば、説明するDOSコールが必要とするパラメータの名前、データサイズ、パラメータの意味などについて書かれています。

パラメータの名前の後のカッコ内の文字について、rはDOSコールによって読まれる引数、wはDOSコールによって書き換えられる可能性のある引数（ポインタの場合はその示す先が書き換えられる）を表します。

・**コール**

説明するDOSコールを呼び出すまでの手続きが書かれています。

ここでの図はコールする直前のスタックの概念図です。

・**リターン**

もしあれば、説明するDOSコールの処理が終了した時点で、戻る値の意味が書かれます。

・**解説**

説明するDOSコールの機能、より詳細に示します。

2.2.4 個々のDOSコール

DOSコール番号とそれぞれの機能、および入力すべきパラメータがあればそのパラメータ、もしあればリターン値などを以下に示します。

章末には、DOSコールをいくつか使用したサンプルプログラムを掲載してあります。

※システムディスクのINCLUDEディレクトリの中にDOSコール用のヘッダファイルとして、

DOSCALL. MAC

というファイルを用意してあります。

このファイルをアセンブラプログラムの先頭でインクルードしておくとDOSコールを行う際、

DC. W　$FF??

とコール番号でコールする代わりに、

DC. W　_EXIT

などのエントリ名でコールできます（エントリ名は大文字）

〈例〉　include DOSCALL. MAC

```
.
.
.
DC.W   _EXIT
```

また、マクロを使用すれば

```
DOS   _EXIT
```

のように書くことができます

〈例〉　include DOSCALL.MAC

```
DOS   _EXIT
```

exit

● exit ( )　　　DOS コール番号：$FF00

機　能	プログラムを終了します。
引　数	なし
コール	DC. W　　_EXIT
リターン	なし
解　説	プログラムを終了します。 現在オープンしているファイルハンドルはクローズします。 子プロセスがオープンしたファイルハンドルもクローズします。 この後に書いたプログラムは実行されません。

getchar

● getchar ( )　　DOS コール番号：$FF01

機　能	キーボードの入力を待ちます（エコーつき）。
引　数	なし
コール	DC. W　_GETCHAR
リターン	D0. L　キーコード
解　説	キーの入力を待って、入力したキーコードを標準出力に表示します。 入力されたキーが“^C”（CTRL+C　ブレーク）、“^P”（CTRL+P　以後の標準出力をプリンタにも出力する）、“^N”（CTRL+N　^P状態をやめる）のいずれかであるかどうかもチェックします。 もしこれらのキーが入力されたら、以下のような動作をします。 　^C……親プロセスに戻る 　^P……プリンタに出力する リターンのD0. Lレジスタには、キーコードが入ります。

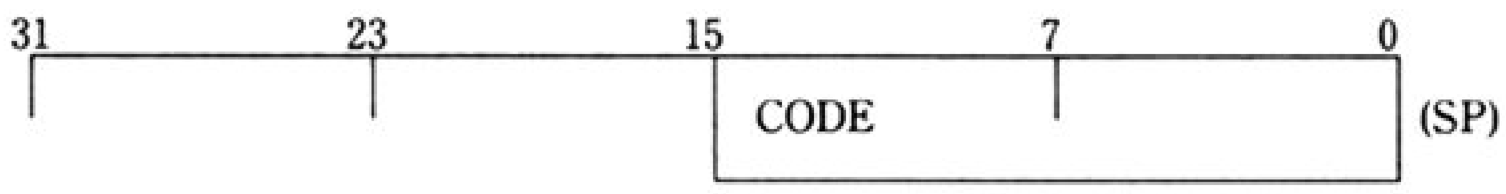
putchar

● putchar (CODE)　　DOS コール番号：$FF02

機　能	文字コードにより文字を表示します。
引　数	CODE (r)　word　1バイト文字コード

コール

```
MOVE. W     CODE, -(SP)
DC. W       _PUTCHAR
ADDQ. L     #2, SP
```



リターン　なし

解　説　CODEで指定した文字コードの文字を、標準出力に表示します。
表示中に入力されたキーが"^C" (CTRL+C　ブレーク)、"^S" (CTRL+S　画面表示の一時停止)、"^P" (CTRL+P　以後の標準出力をプリンタにも出力する)、"^N" (CTRL+N　^P状態をやめる) のいずれかであるかどうかもチェックします。
もしこれらのキーが入力されたら、以下のような動作をします。
　^C……親プロセスに戻る
　^P……プリンタに出力する

cominp

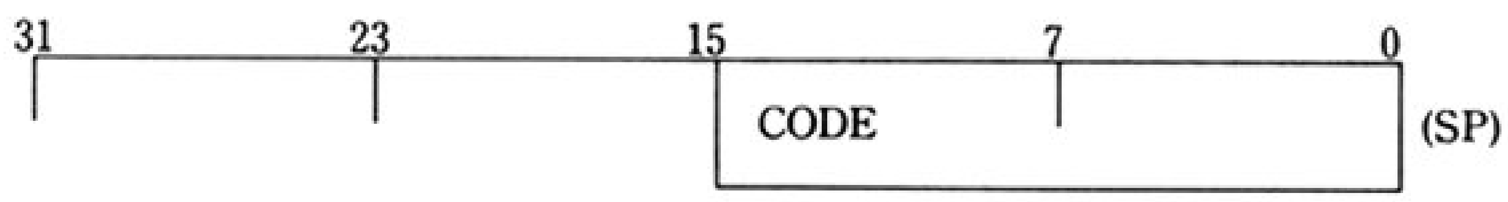
● cominp ( )　　DOS コール番号：$FF03

機　能	RS-232C 回線から 1 バイト入力します。
引　数	なし
コール	DC. W　_COMINP
リターン	D0. L　入力した 1 バイト
解　説	RS-232C 回線から 1 バイトを入力します。 入力の際、"^C"についてブレークチェックします。 リターンの D0. L レジスタに、RS-232C 回線から入力した 1 バイトを返します。

comout

● comout (CODE)　　　DOS コール番号：$FF04

機　能	RS-232C 回線に 1 バイト出力します
引　数	CODE (r)　　word　　1 バイトコード
コール	MOVE. W　　CODE, −(SP) DC. W　　_COMOUT ADDQ. L　　#2, SP



リターン	なし
解　説	CODE で指定した 1 バイトのコードを、RS-232C 回線へ出力します。 このとき、“^C”についてブレークチェックをします。

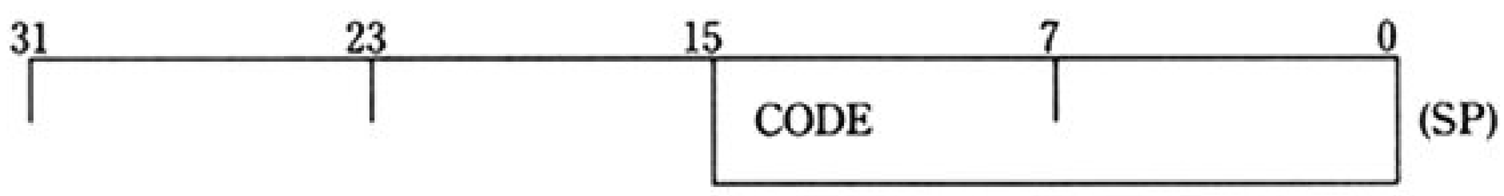
prnout

● prnout (CODE)　　DOS コール番号：$FF05

機　能	プリンタに 1 文字出力します
引　数	CODE (r)　　word　　1 バイト文字コード

コール

```
MOVE. W     CODE, −(SP)
DC. W       _PRNOUT
ADDQ. L     #2, SP
```



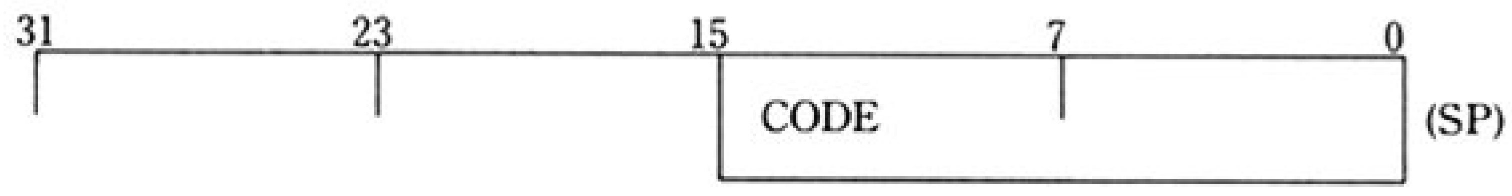
リターン　なし

解　説　CODE で指定した 1 文字を、プリンタへ出力します。
このとき、"^C"についてブレークチェックをします。
CODE の上位バイトは 0 でなければならないので漢字は上位、下位の順で 1 バイトずつ出力してください。

inpout

● inpout (CODE)　　DOS コール番号：$FF06

機　能	キーの入出力をします。
引　数	CODE (r)　　word　　1 バイト文字コード
コール	MOVE. W　　CODE, −(SP) DC. W　　_INPOUT ADDQ. L　　# 2, SP



リターン	CODE ＝ $FF, $FE： D0. L　　入力されたキーコード（0 なら入力なし） CODE ＝ 上記以外： D0. L　　0
解　説	CODE の値によって、次の処理をします。 CODE ＝ $FF……キー入力する（入力がなくても待たない） CODE ＝ $FE……キー入力を調べる（入力キーコードを返す） CODE ＝ 上記以外文字コードとして出力 ブレークチェックはしません。 リターンの D0. L レジスタには CODE が$FF か$FE の場合、入力されたキーコードを返し、それ以外の場合は 0 を返します。 CODE が$FF か$FE でリターン値(D0. L)が 0 ならば、入力がなかったことを表します。

inkey

● inkey ( )　　DOS コール番号：$FF07

機　能	キーボードから 1 文字入力します（ブレークチェックなし）。
引　数	なし
コール	DC. W　_INKEY
リターン	D0. L　入力されたキーのキーコード
解　説	キーが押されるまで待ち、そのキーコードを返します。 ブレークチェックはしません。 ブレークチェックをしたい場合は、getc（$FF08）を使用してください。 リターンの D0. L レジスタには、入力されたキーコードを返します。

getc

● getc ( )　　　DOS コール：$FF08

機　能	キーボードから1文字入力します（ブレークチェックあり)。
引　数	なし
コール	DC. W　　_GETC
リターン	D0. L　　入力されたキーのキーコード
解　説	キーが押されるまで待ち、そのキーコードを返します。 “^C”、“^P”、“^N”についてチェックします。 リターンのD0. Lレジスタには、押されたキーコードを返します。

print

● print (MESPTR)　　DOS コール番号：$FF09

機　　能　文字列を表示します。

引　　数　MESPTR (r)　address　文字列のポインタ

コ ー ル

```
          PEA       MESPTR
          DC. W     _PRINT
          ADDQ. L   # 4, SP
             :
MESPTR:   DC. B     'TESTSAMPLE', 13, 10, 0
```

31	23	15	7	0	
MESPTR					(SP)

リターン　なし

解　　説　MESPTR で指定したポインタ以降、ヌル文字(0)までの文字列を表示します。
文字列のポインタ MESPTR はスタックに入れて指定します。
“^C”、“^S”、“^P”、“^N”についてチェックをします。

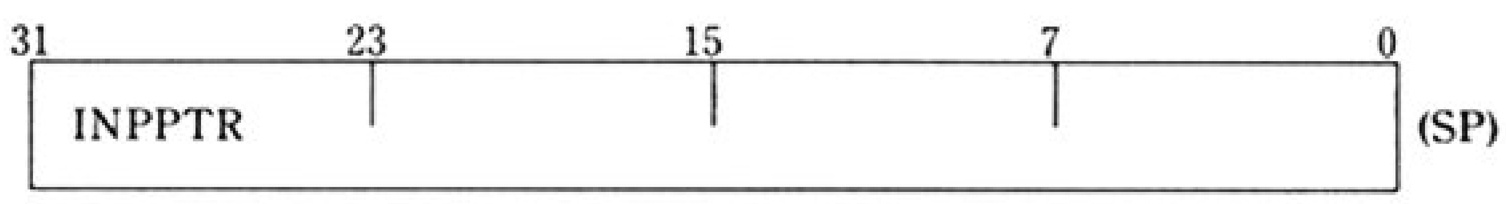
gets

● gets (INPPTR)　　DOS コール番号：$FF0A

機　能	文字列を入力します。
引　数	INPPTR (w)　address　入力バッファのポインタ

コール

```
          PEA       INPPTR
          DC. W     _GETS
          ADDQ. L   #4, SP
              :
INPPTR:   DC. B     80
          DC. B     5
          DS. B     81
```



リターン	D0. L　入力文字数

解　説

入力バッファをINPPTRで指定して、改行（リターンキー）までの入力文字列をバッファに書き込みます。

改行文字は、バッファに書き込む前にヌル文字(0)に置き換えます。

“^C”、“^P”、“^N”についてチェックをします。

入力バッファINPPTRは、次のように設定します。

```
INPPTR:     DC. B   n     *入力最大文字数
INPPTR+1:   DC. B   x     *実際に入力した文字数が入るワーク
INPPTR+2:   DS. B   n+1   *入力最大文字数+1のデータエリアを確保
```

入力最大文字数を超えた場合は、警告を出しますが終了はしません。

リターンのD0. Lレジスタには、INPPTR+1と同じ入力文字数を返します。

これはINPPTR+2からヌル文字(0；文字列の終端)までのバイト数にあたります。

keysns

● keysns ( )　　DOS コール番号：$FF0B

機　能	キーの入力状態をチェックします。
引　数	なし
コール	DC. W　　_KEYSNS
リターン	D0. L　　入力状態 D0. L ＝ 0 …入力なし D0. L ＝ −1 入力あり
解　説	キーの入力状態を調べます。 “^C”、“^P”、“^N”についてチェックをします。

kfluch

kflush

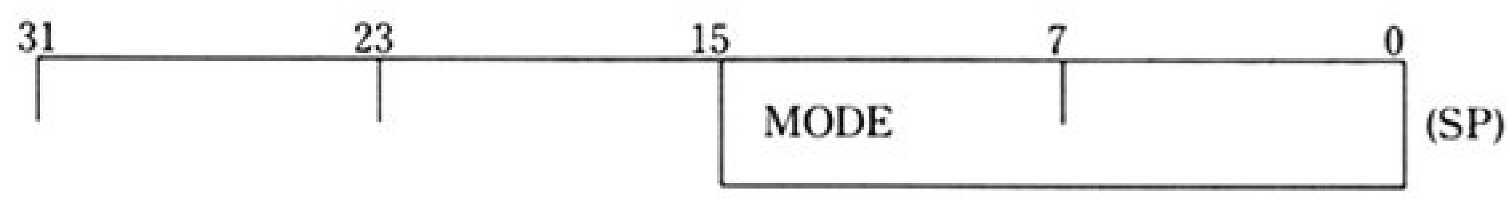
● kflush (MODE, INPPTR)　　DOS コール番号：$FF0C

機　　能	バッファフラッシュして（空にして）からキーボードから入力します。
引　　数	MODE (r)　word　　モード CODE (r)　word　　処理を選択するコード INPPTR (w)address　入力バッファのポインタ

コ　ー　ル

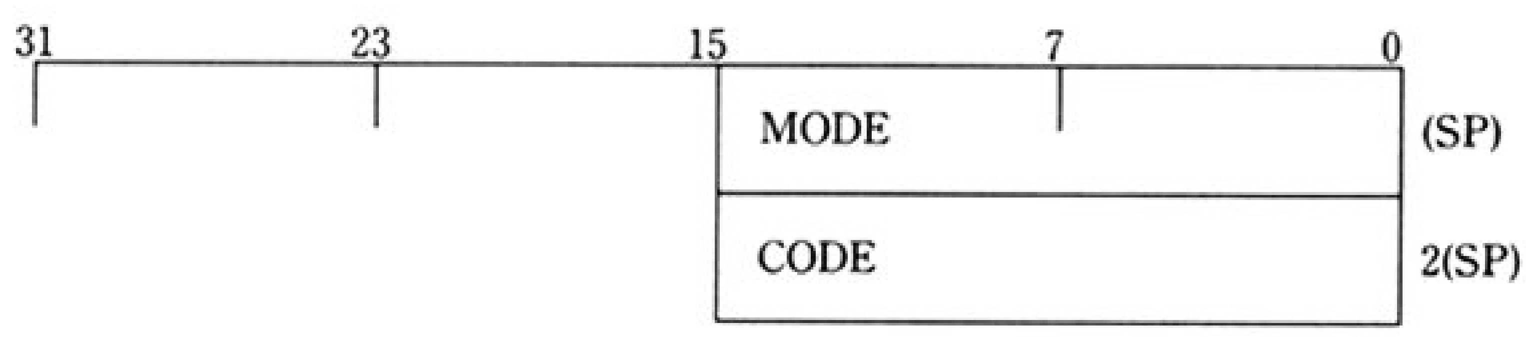
〈MODE ＝ $01, $07, $08〉

```
        MOVE. W   MODE, −(SP)
        DC. W     _KFLUSH
        ADDQ. L   #2, SP
```



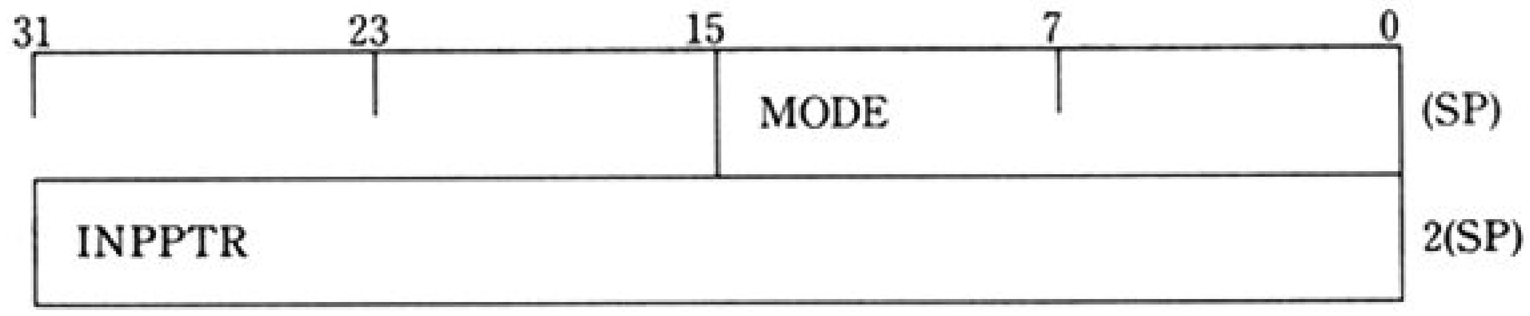
〈MODE ＝ $06〉

```
        MOVE. W   CODE, −(SP)
        MOVE, W   MODE, −(SP)
        DC. W     _KFLUSH
        ADDQ. L   #4, SP
```



〈MODE ＝ $0A〉

```
        PEA       INPPTR
        MOVE. W   MODE, −(SP)
        DC. W     _KFLUSH
        ADDQ. L   #6, SP
          ⋮
INPPTR: DC. B     80
        DC. B     0
        DS. B     81
```



リターン

〈MODE ＝ $01, $07, $08〉

D0. L　　入力されたキーコード

〈MODE ＝ $06〉

CODE ＝ $FE, $FF：

D0. L　　入力されたキーコード（0なら入力なし）

CODE ＝ 上記以外：

D0. L　　0

〈MODE ＝ $0A〉

D0. L　　入力文字数

解　　説

キー入力バッファをフラッシュして（空にして）から、キーボードより入力を行います。

MODEの値は、$0?がそれぞれDOSコールの$FF0?に対応しており、以下のような動作をします。

MODE ＝ $01　kflush ($01)

キーの入力を待ち、入力されたコードを標準出力に出力（エコーつき）し、そのキーコードをD0. Lに返します。

MODE ＝ $06　kflush ($06, CODE)

CODE ＝ $FF……キーの入力（入力が無くても待たない）

CODE ＝ $FE……キーセンス（入力キーコードを返す）

CODE ＝ 上記以外……標準出力に出力

MODE ＝ $07　kflush ($07)

キー入力を待ち、そのキーコードをD0. Lに返します（ブレークチェックなし）

MODE ＝ $08　kflush ($08)

キー入力を待ち、そのキーコードをD0. Lに返します（ブレークチェックあり）

MODE ＝ $0A　kflush ($0A, INPPTR)

改行（リターン）までに入力した文字列をバッファに書き込みます。

入力バッファ INPPTR は、次のように設定します。

INPPTR：	DC. B	n	＊入力最大文字数
INPPTR+1：	DC. B	x	＊実際に入力した文字数が入るワーク
INPPTR+2：	DC. B	n+1	＊入力最大文字数+1のデータエリアを確保

リターンの D0. L には、INPPTR+1 と同じ入力文字数を返します。
これは INPPTR+2 からヌル文字（0；文字列の終端）までのバイト数にあたります。

fflush

● fflush ( )　　DOS コール番号：$FF0D

機　能	ディスクをリセットします。
引　数	なし
コール	DC. W　_FFLUSH
リターン	なし
解　説	ディスクのリセットを行います。 ディスクバッファの内容はすべてフラッシュしますが、ファイルのクローズはしません。

chgdrv

● chgdrv (DRIVE)　　DOS コール番号：$FF0E

機　能	カレントドライブを指定します。
引　数	DRIVE (r)　word　ドライブ番号
コール	MOVE. W　DRIVE, −(SP) DC. W　_CHGDRV ADDQ. L　#2, SP

31–16	15–0
	DRIVE

(SP)

リターン	D0. L　指定可能ドライブ数 (1〜N)
解　説	カレントドライブを変更します。

変更先のドライブ番号は、DRIVE を使って次のように指定します。

DRIVE ＝ 0……A ドライブ
DRIVE ＝ 1……B ドライブ
⋮　⋮

以下、2、3、4……は、C. D. E となります。

リターンの D0. L には、指定可能なドライブ数 (1〜N) を返します。
ただし、指定した値以下の値が返されたときはエラーを意味します。

drvctrl

● drvctrl (MODE[,P1, …Pn])　　DOSコール番号：$FF0F

機　　能　ドライブの状態をチェック／設定します。

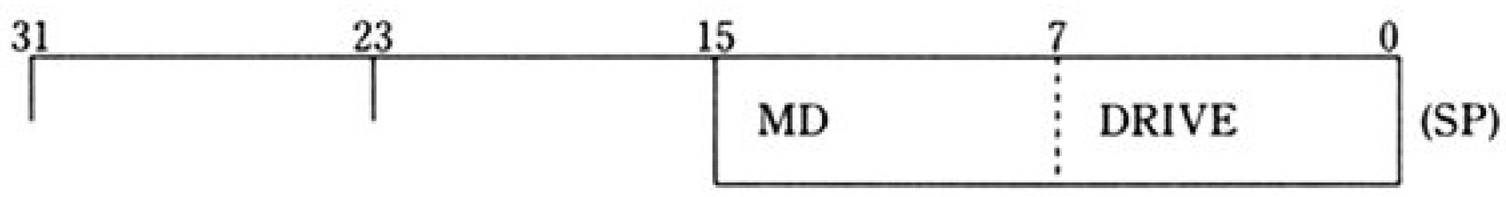
引　　数

MODE	word	モードおよびドライブ番号MD×256＋DRIVE
P1	long	引数1
P2	long	引数2
	⋮	
Pn	long	引数n

コ ー ル

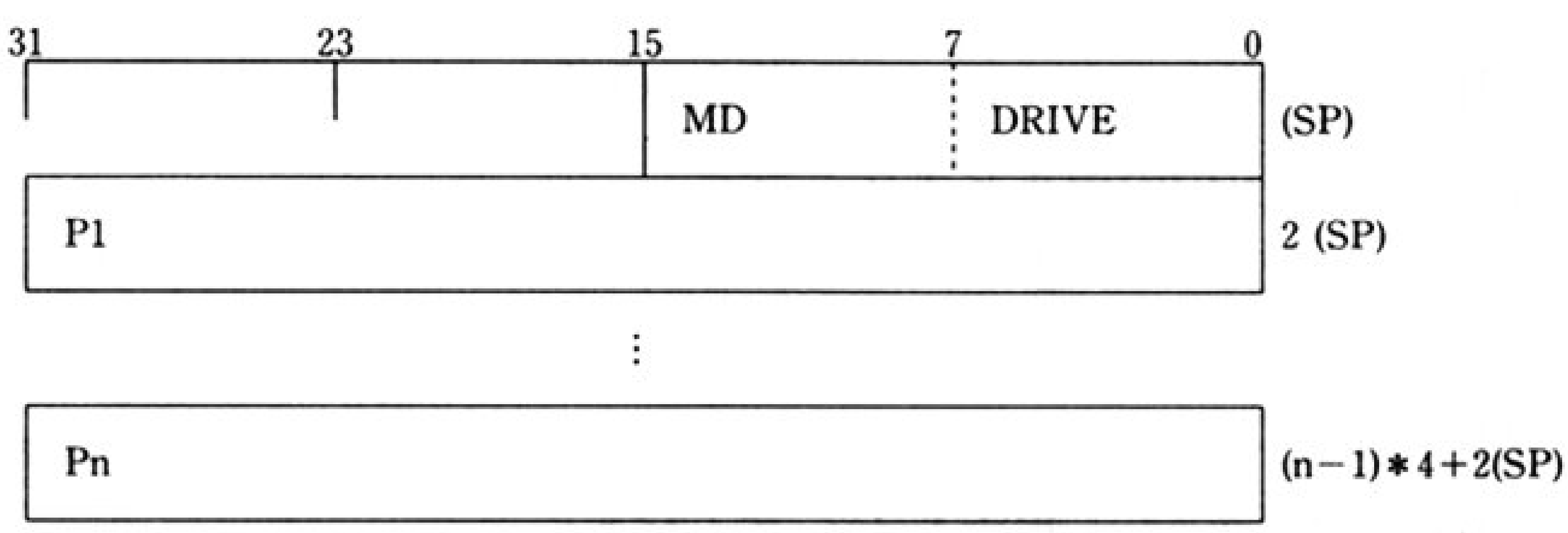
<MD ＝ 0～4, 9, 10>

```
MOVE. W   MODE, -(SP)
DC. W     _DRVCTRL
ADDQ. L   #2, SP
```



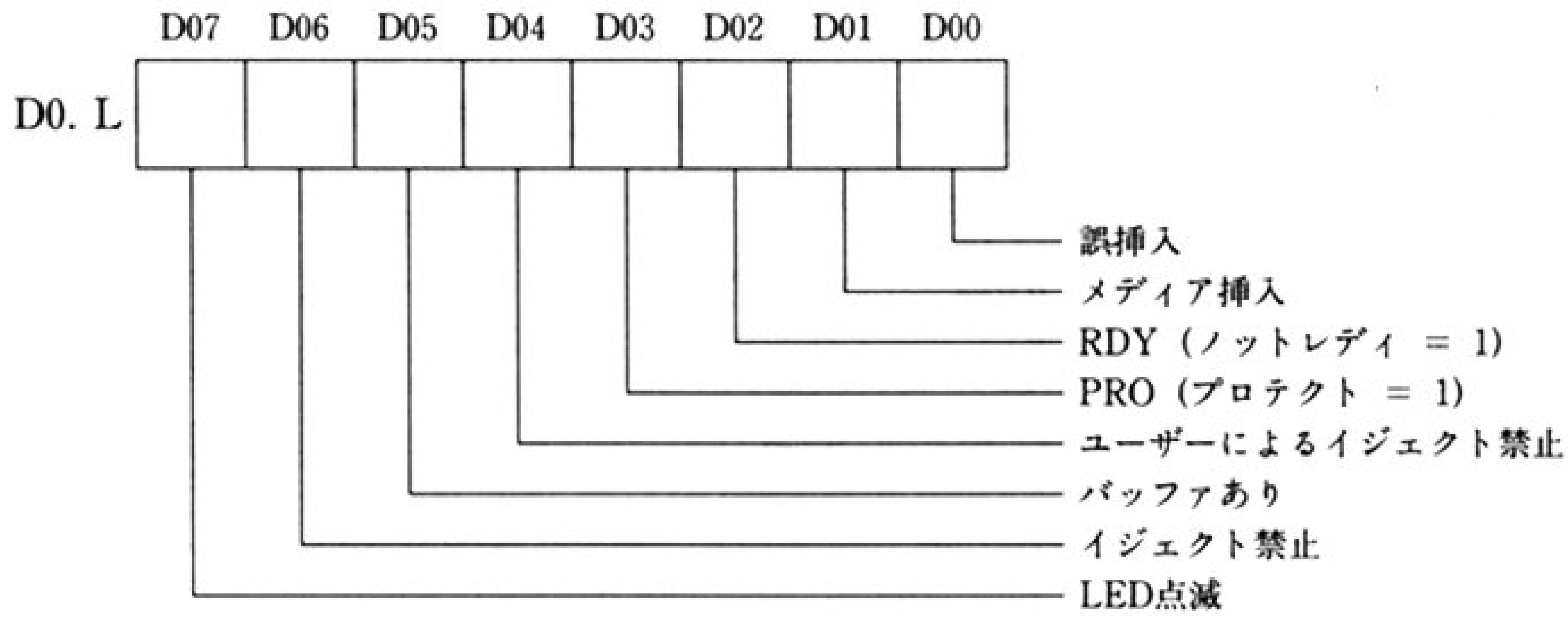
<MD ＝ 16～??>

```
MOVE. L   Pn, -(SP)                    *引数n
          ⋮
MOVE. L   P1, -(SP)                    *引数1
MOVE. W   MODE, -(SP)
DC. W     _DRVCTRL
LEA       n*4+2 (SP), SP               *引数の数により変わります
```



リターン　〈MD ＝ 0〉

D0. L　　指定ドライブの状態



注：PRO（プロテクト ＝ 1）とRDY（ノットレディ ＝ 1）は、メディアが挿入されていた場合のみ返します。

〈MD ＝ 1～??〉

なし

解　　説　ドライブ番号で指定したドライブの状態を、モードの値にしたがってチェックあるいは設定します。

MODEの上位8ビットにモードMDを、下位8ビットにドライブ番号DRIVEを指定します。

モードMDの内容は次の通りです。

MD ＝ 0 …………状態チェック

MD ＝ 1 …………イジェクト

イジェクトしようとしたドライブに、オープンされているファイルがあるとエラーです。

MD ＝ 2 …………イジェクト禁止（MD ＝ 1のイジェクトも禁止）

MD ＝ 3 …………イジェクト許可

オープンされているドライブはクローズされず、バッファは自動的にフラッシュ（消去）されます。

MD ＝ 4 …………ディスクがセットされていないときにLED点滅

MD ＝ 5～8 ……指定しないでください。

MD ＝ 9 …………指定したドライブにオープンされているファイルがあるとエラーです。

カレントディレクトリはルートディレクトリとなり、サーチFATも先頭からになります。

MD = 10 ………指定したドライブのサーチFATが先頭からになります。

MD = 11～15 …予約されています。

MD = 16～??……特殊ブロックデバイスに対して用います。

この場合、任意の引数を追加できます。

ドライブ番号DRIVEは次のように指定します。

DRIVE = 0 ……カレントドライブ

DRIVE = 1 ……Aドライブ

DRIVE = 2 ……Bドライブ

⋮　　⋮

consns

● consns ( )　　　DOS コール番号：$FF10

機　　能	画面出力ができるかどうかをチェックします。
引　　数	なし
コ ー ル	DC. W　　_CONSNS
リターン	D0. L　　出力状態 D0. L ＝ 0………出力不可 D0. L ≠ 0………出力可能
解　　説	画面へ出力可能か否かを調べます。

prnsns

● prnsns ( )　　DOS コール番号：$FF11

機　能	プリンタ出力ができるかどうかをチェックします。
引　数	なし
コール	D0. W　_PRNSNS
リターン	D0. L　出力状態 D0. L ＝ 0………出力不可 D0. L ≒ 0………出力可能
解　説	プリンタへ出力可能か否かを調べます。

cinsns

●cinsns ( )　　DOSコール番号：$FF12

機　　能	RS-232C回線から入力ができるかどうかをチェックします。
引　　数	なし
コ ー ル	DC. W　_CINSNS
リターン	D0. L　入力状態 D0. L ＝ 0………入力不可 D0. L ≠ 0………入力可能
解　　説	RS-232C回線からの入力が可能か否かを調べます。

coutsns

● coutsns ( )　　DOSコール番号：$FF13

機　　能	RS-232C回線へ出力ができるかどうかをチェックします。
引　　数	なし
コ ー ル	DC. W　　_COUTSNS
リターン	D0. L　　出力状態 D0. L ＝ 0………出力不可 D0. L ≠ 0………出力可能
解　　説	RS-232C回線への出力が可能か否かを調べます。

fatchk

●fatchk (FILE, BUFFER [+$80000000, LEN])　　DOSコール番号：$FF17

機　　能　ドライブのセクタが連続しているかどうかをチェックします。

引　　数

FILE (r)	address	文字列（ファイル名）のポインタ
BUFFER (w)	address	バッファのポインタ
LEN (r)	word	バッファのバイト数

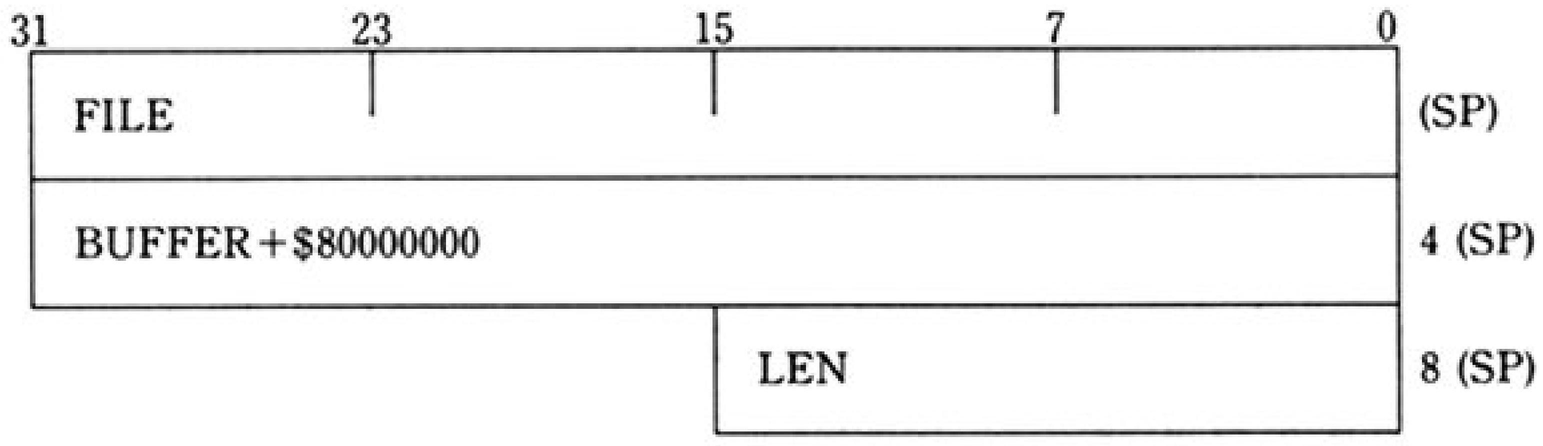
コ ー ル　①

```
            PEA       BUFFER
            PEA       FILE
            DC. W     _FATCHK
            ADDQ. L   #8, SP
                 ⋮
FILE:       DC. B     'HUMAN. SYS', 0
BUFFER:     DS. B     ???     *???はバッファのバイト数
```

31 ... 0	
FILE	(SP)
BUFFER	4 (SP)

②

```
            MOVE. W   LEN, -(SP)
            PEA       BUFFER+$80000000
            PEA       FILE
            DC. W     _FATCHK
            LEA       10 (SP), SP
                 ⋮
FILE:       DC. B     'HUMAN. SYS', 0
BUFFER:     DS. B     ???       *???はバッファのバイト数
```



リターン　D0. L　バッファの使用バイト数（終わりの0を含む）

D0. L ＝ 8 …連続（①でコールした場合）

D0. L ＝ 14 …連続（②でコールした場合）

D0. L ＜ 0 …エラーコード

解　説　指定ファイルのドライブ番号と、セクタの連続性をバッファに書き込みます。

このDOSコールには2種類のコール方法があります。

それは、バッファのアドレスBUFFERを引数として渡すとき、

①そのままスタックに積む

②BUFFER＋$80000000をスタックに積む

の2種類です。

①の場合はバッファに書き込まれるセクタ番号とセクタ数のサイズがワードですが、②の場合はロングワードとなります。

なお、②の場合、バッファのバイト数LENが引数として必要であることに注意してください。

以下にバッファに書き込まれる情報を示します。

```
①の場合  DC. W   d    *ドライブ番号
          DC. W   s1   *先頭セクタ番号
          DC. W   n1   *セクタ数
          DC. W   s2   *次のセクタ番号
          DC. W   n2   *セクタ数
               ⋮
          DC. W   0    *終わりを表す0

②の場合  DC. W   d    *ドライブ番号
          DC. W   s1   *先頭セクタ番号
          DC. L   n1   *セクタ数
          DC. L   s2   *次のセクタ番号
          DC. L   n2   *セクタ数
```

```
            ⋮
        DC. L   0       *終わりを表す0
```

ドライブ番号dとドライブ名の対応は次の通りです。

d = 1 ……Aドライブ
d = 2 ……Bドライブ
d = 3 ……Cドライブ
　⋮　　　　⋮

このDOSコールでFATが連続していることを調べるとともに、diskred($FFF3)でデータ部を直接読むこともできます。

リターンのD0. Lには、バッファの使用バイト数（終わりの0を含む）を返します。

①の場合、リターン値が8なら連続です。

②の場合、リターン値が14なら連続です。

負の数ならエラーコードです（エラーコードは2. 1. 7を参照）。

hendsp

● hendsp (MD［, POS］［, MESPTR］)　　DOS コール番号：$FF18

機　能　漢字変換行をコントロールします。

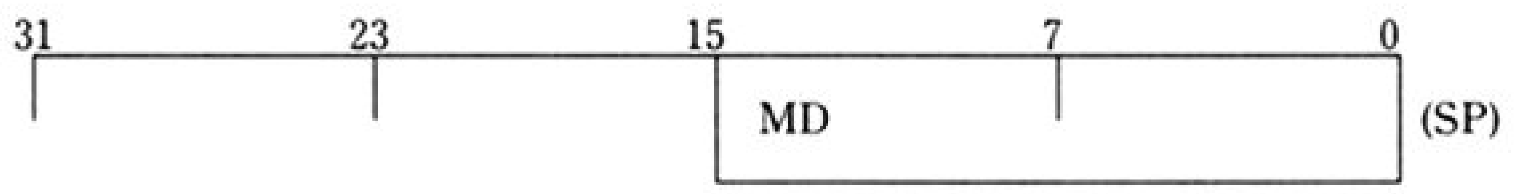
引　数

MD (r)	word	モード
POS (r)	word	ポジション(変換ウィンドウの先頭からの位置)
MESPTR (r)	address	変換文字列のポインタ

コール

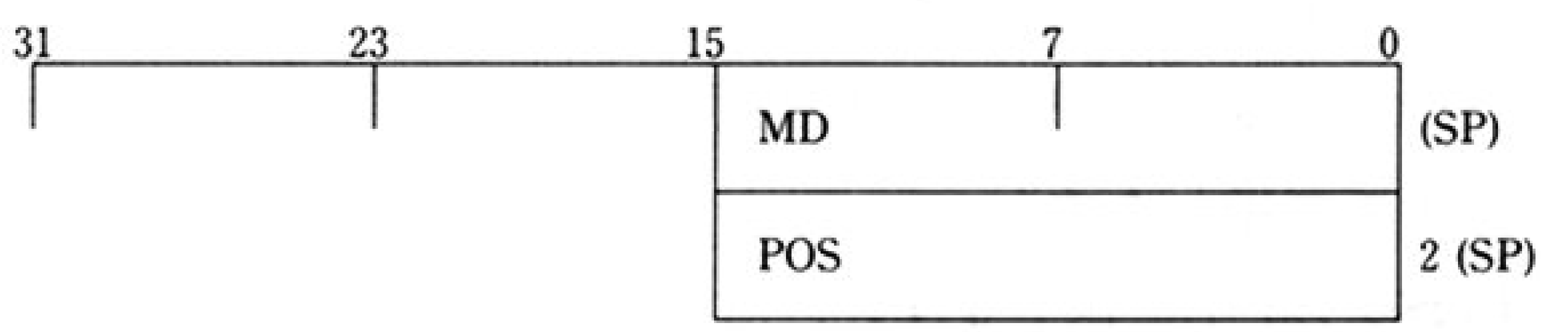
〈MD ＝ 0, 3, 4, 8, 11〉

```
        MOVE. W     MD, -(SP)
        DC. W       _HENDSP
        ADDQ. L     #2, SP
```



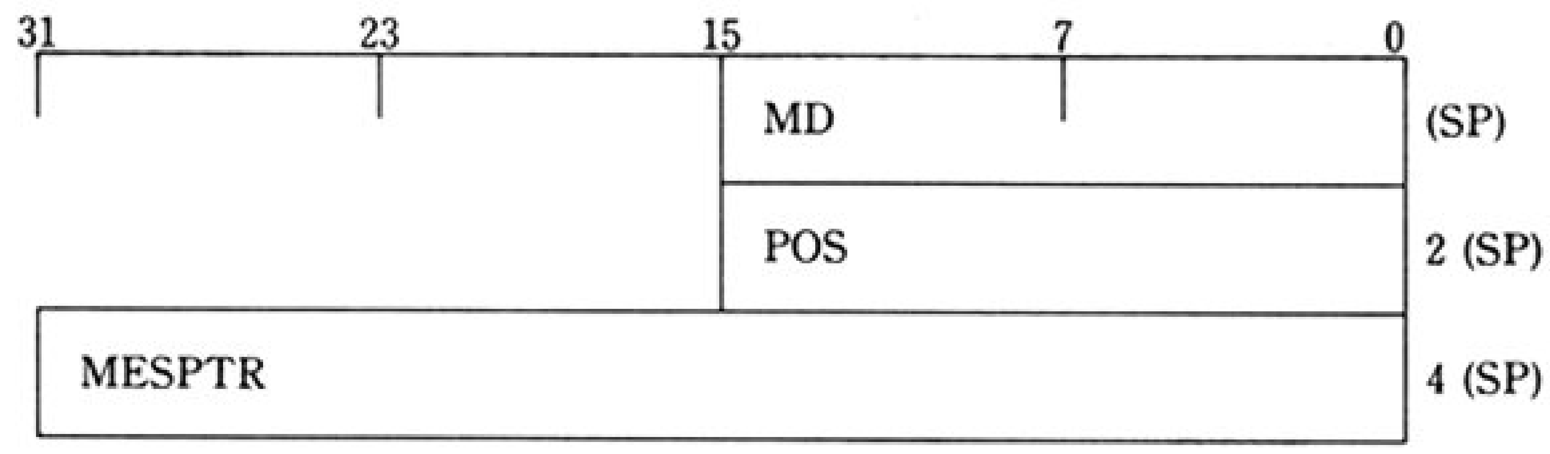
〈MD ＝ 7〉

```
        MOVE. W     POS, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _HENDSP
        ADDQ. L     #4, SP
```



〈MD ＝ 1, 2, 5, 6, 9, 10〉

```
        PEA         MESPTR
        MOVE. W     POS, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _HENDSP
        ADDQ. L     #8, SP
              :
MESPTR: DC. B       'SHARP', 13, 10, 0
```



リターン

〈MD ＝ 0, 4, 8〉

D0, L　　　ウィンドウの最大文字数

〈MD ＝ 1, 2, 5, 6, 9, 10〉

D0. L　　　次のポジション

〈MD ＝ 3, 7, 11〉

なし

解　説

漢字変換行をコントロールします。
このDOSコールは、漢字変換を扱うアプリケーションプログラムに使用します。ただし、日本語FP（かな漢字変換フロントプロセッサ）から使用するので、一般のアプリケーションが使用してはいけません。
MDの内容と意味を次に示します。

MD ＝ 0　hendsp (0)
　モード表示ウィンドウをオープンします。
　最大文字数をD0. Lに返します。

MD ＝ 1　hendsp (1, POS, MESPTR)
　ノーマルで表示します。
　次のPOSをD0. Lに返します。

MD ＝ 2　hendsp (2, POS, MESPTR)
　リバースで表示します。
　次のPOSをD0. Lに返します。

MD ＝ 3 hendsp (3)
　モード表示ウィンドウをクローズします。

MD ＝ 4　hendsp (4)
　入力ウィンドウをオープンします。
　最大文字数をD0. Lに返します。

MD ＝ 5　hendsp (5, POS, MESPTR)
　ノーマルで表示します。
　次のPOSをD0. Lに返します。

MD = 6　hendsp (6, POS, MESPTR)

リバースで表示します。

次のPOSをD0. Lに返します。

MD = 7　hendsp (7, POS)

POS以後をもとに戻します。

MD = 8　hendsp (8)

候補ウィンドウをオープンします。

最大文字数をD0. Lに返します。

MD = 9　hendsp (9, POS, MENPTR)

ノーマルで表示します。

次のPOSをD0. Lに返します。

MD = 10　hendsp (10, POS, MESPTR)

リバースで表示します。

次のPOSをD0. Lに返します。

MD = 11　hendsp (11)

候補ウィンドウをクローズします。

curdrv

●curdrv ( )　　DOSコール番号：$FF19

機　能	カレントドライブの値を返します。
引　数	なし
コール	DC. W　_CURDRV
リターン	D0. L　カレントドライブの番号 D0. L ＝ 0 ……Aドライブ D0. L ＝ 1 ……Bドライブ D0. L ＝ 2 ……Cドライブ ⋮　⋮
解　説	カレントドライブの値を返します。

getss

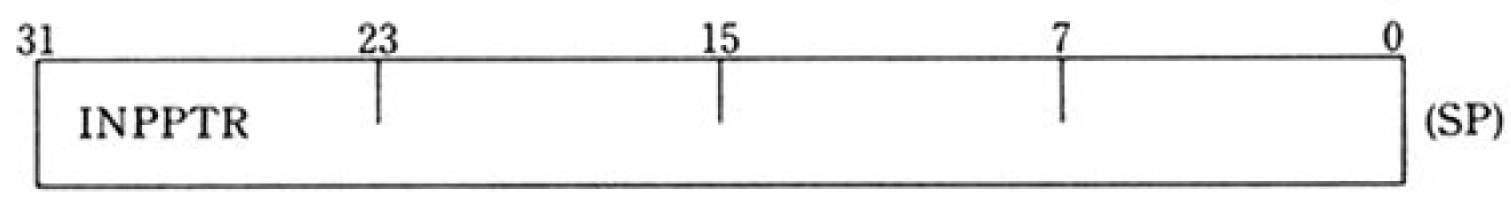
● getss (INPPTR)　　DOSコール：$FF1A

機　能　文字列を入力します。

引　数　INPPTR (w)　address　入力バッファのポインタ

コール

```
        PEA      INPPTR
        DC. W    _GETSS
        ADDQ. L  #4, SP
           ⋮
INPPTR: DC. B    80
        DC. B    0
        DS. B    81
```



リターン　D0. L　入力文字数（INPPTR＋1と同じ）

解　説　入力バッファをINPPTRで指定して、改行（リターンキー）までの入力文字列をバッファに書き込みます。

改行文字は、バッファに書き込む前にヌル文字（0）に置き換えます。

ブレークチェックはしません（VOID／NEWLINEで改行しません）。

入力最大文字数を超えた場合は、警告を出しますが終了はしません。

入力バッファINPPTRは、次のように設定します。

INPPTR:	DC. B	n	＊入力最大文字数
INPPTR+1	DC. B	x	＊実際に入力した文字数が入るワーク
INPPTR+2	DS. B	n+1	＊入力最大文字数の+1のデータエリアを確保

リターンのD0. Lレジスタには、INPPTR＋1と同じ入力文字数を返します。

これはINPPTR＋2から、ヌル文字（0；文字列の終端）までのバイト数にあたります。

fgetc

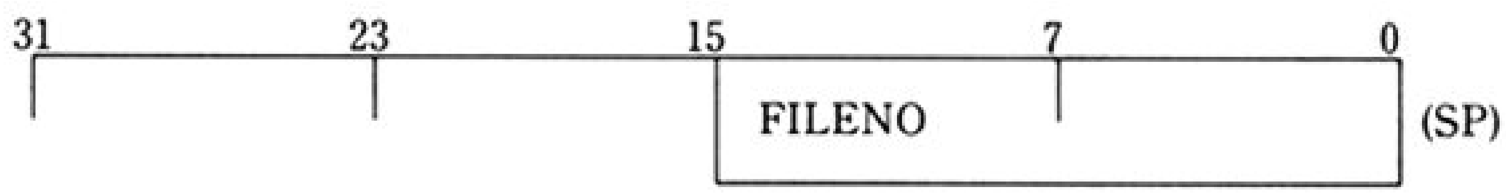
● fgetc (FILENO)　　DOS コール番号：$FF1B

機　　能　ファイルハンドルから1バイト入力します。

引　　数　FILENO (r)　word　ファイルハンドル

コ　ー　ル

```
MOVE. W    FILENO, -(SP)
DC. W      _FGETC
ADDQ. L    #2, SP
```



リターン　D0. L　入力された文字の文字コード

解　　説　FILENOで指定したファイルハンドルから入力があるまで待ち、入力があったらD0. Lレジスタにそのコードを返します。

fgets

● fgets (INPPTR, FILENO)　　　DOS コール番号：$FF1C

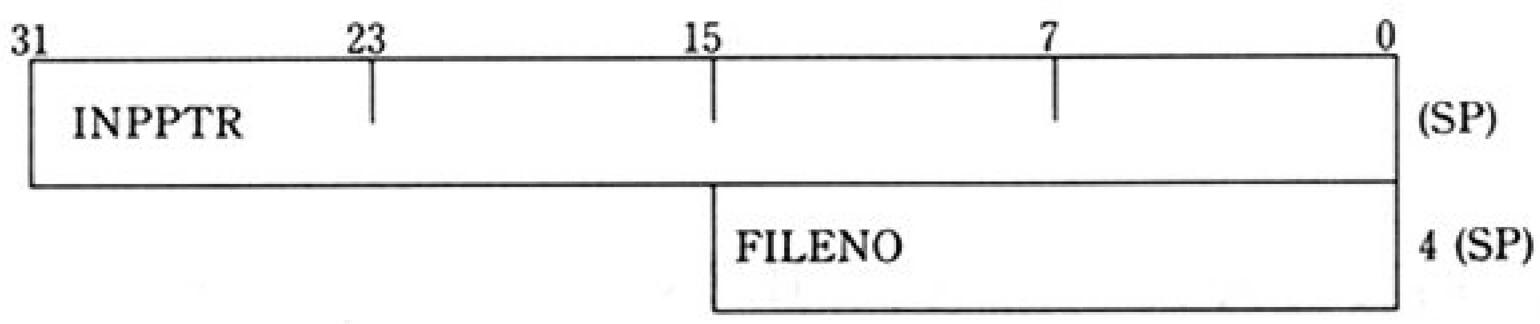
機　能　ファイルハンドルから文字列を入力します。

引　数

INPPTR (w)	address	入力バッファのポインタ
FILENO (r)	word	ファイルハンドル

コール

```
         MOVE. W   FILENO, -(SP)
         PEA       INPPTR
         DC. W     _FGETS
         ADDQ. L   #6, SP
           ⋮
INPPTR:  DC. B     80
         DC. B     0
         DS. B     81
```



リターン　D0. L　　入力文字数（INPPTR+1 と同じ）

解　説　FILENO で指定したファイルハンドルから、改行コード（キャリッジリターン）までの入力文字列を、INPPTR で指定した入力バッファに書き込みます。
改行文字は、バッファに書き込む前にヌル文字（0）に置き換えます。
ブレークチェックはしません（VOID/NEWLINE で改行しません）。
入力最大文字数を超える場合も、入力最大文字数まで入れて終了します。
入力バッファ INPPTR は、次のように設定します。

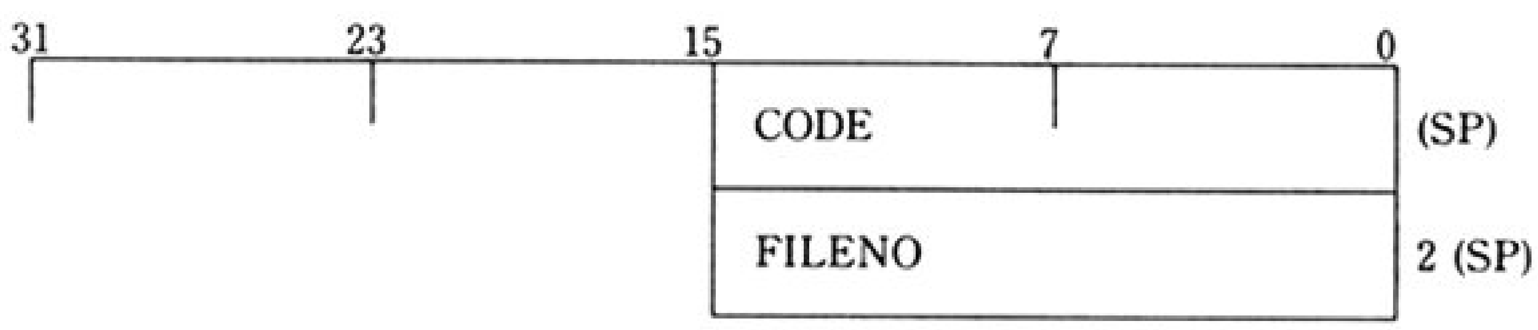
INPPTR:	DC. B	n	＊入力最大文字数
INPPTR+1:	DC. B	x	＊実際に入力した文字数が入るワーク
INPPTR+2	DS. B	n+1	＊入力最大文字数+1 のデータエリアを確保

リターンの D0. L レジスタには、INPPTR+1 と同じ入力文字数を返します。
これは INPPTR+2 から、ヌル文字（0；文字列の終端）までのバイト数にあたります。

fputc

● fputc (CODE, FILENO)　　　DOS コール番号：$FF1D

機　　能	ファイルハンドルに1文字出力します。
引　　数	CODE (r)　word　1バイト文字コード FILENO (r)　word　ファイルハンドル
コ ー ル	MOVE. W　FILENO, −(SP) MOVE. W　CODE, −(SP) DC. W　_FPUTC ADDQ. L　# 4, SP



リターン	なし
解　　説	CODEで指定した文字コードを、FILENOで指定したファイルハンドルに出力します。 FILENOがキャラクタデバイスの場合には、“^C”、“^S”、“^P”、“^N”についてチェックをします。

fputs

● fputs (MESPTR, FILENO)　　DOS コール番号：$FF1E

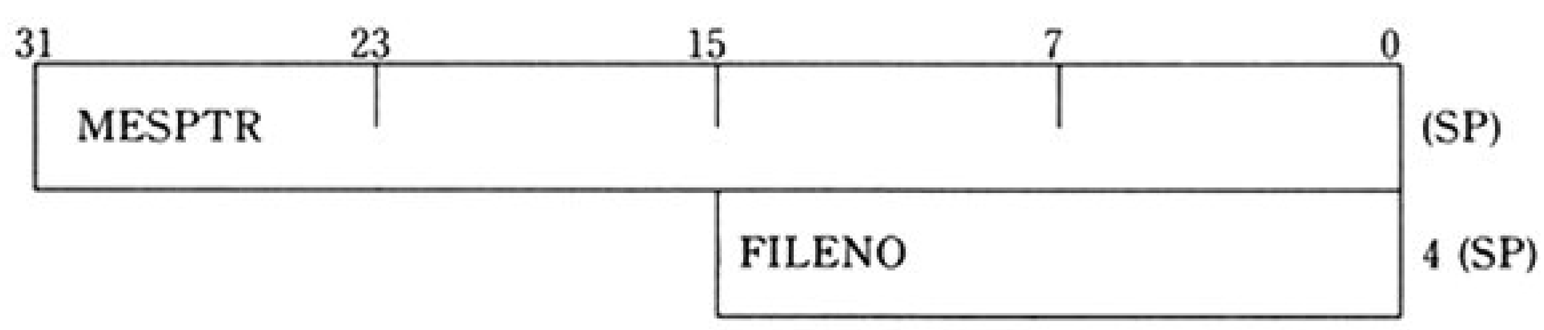
機　能　ファイルハンドルに文字列を出力します。

引　数

MESPTR (r)	address	文字列のポインタ
FILENO (r)	word	ファイルハンドル

コール

```
            MOVE. W    FILENO, -(SP)
            PEA        MESPTR
            DC. W      _FPUTS
            ADDQ. L    #6, SP
              :
MESPTR :    DC. B      'SHARP', 13, 10, 0
```



リターン　なし

解　説　MESPTR で指定したポインタからの文字列を、FILENO で指定したファイルハンドルに出力します。

文字列の末尾のヌル文字（0；文字列の終端）は出力しません。

FILENO がキャラクタデバイスの場合には、“^C”、“^S”、“^P”、“^N”、についてチェックをします。

allclose

● allclose ( )　　DOS コール番号：$FF1F

機　　能	すべてのファイルをクローズします。
引　　数	なし
コ ー ル	DC. W　_ALLCLOSE
リターン	なし
解　　説	現在オープンしているファイルハンドルをクローズします。 子プロセスがオープンしたファイルハンドルもクローズします。 親プロセスがオープンしたファイルハンドルはクローズしません。

super

● super (STACK)　　DOS コール番号：$FF20

機　能　スーパーバイザモード／ユーザーモードを切り替えます。

引　数　STACK (r)　address　切り替えモード

コール

```
            LEA       USERSP, SP
              ⋮
            CLR. L    -(SP)           * スーパーバイザモードへ
            DC. W     _SUPER
            ADDQ. L   #4, SP
            MOVE. L   D0, SSPBUF
            MOVE. L   USP, A0
            MOVE. L   A0, USPBUF
              ⋮                       * スーパーバイザモード
            MOVE. L   USPBUF, A0
            MOVE. L   A0, USP
            MOVE. L   SSPBUF, -(SP)   * ユーザーモードへ
            DC. W     _SUPER
            ADDQ. L   #4, SP
              ⋮
USPBUF:     DS. L     1               * USP BUFFER
SSPBUF:     DS. L     1               * SSP BUFFER
            DS. L     256             * ユーザースタックエリア
USERSP:
```

31–24	23–16	15–8	7–0	
STACK				(SP)

リターン　STACK ＝ 0：

D0. L　　前の SSP の値

D0. L ＜ 0………エラーコード

STACK ≠ 0：

なし

解　　説　スーパーバイザモード／ユーザーモードを切り替えます。

切り替えモードSTACKの内容は次の通りです。

STACK ＝ 0…USPの値をSSPにセットし、スーパバイザーモードに切り替え

STACK ≠ 0…STACKをSSPにセットし、ユーザーモードに切り替え

リターンのD0.Lレジスタは、STACK ＝ 0の場合、前のSSPの値が返されます。

リターンのDO.Lが負の数ならエラーコードです。

fnckey

● fnckey (MODE, BUFFER)　　　DOS コール番号：$FF21

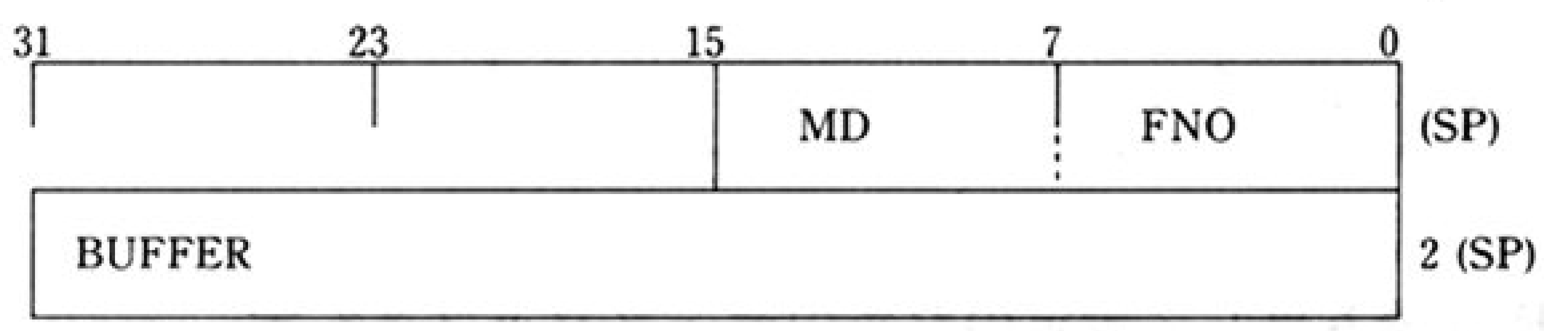
機　　能　再定義可能なキーを読み出し／設定します。

引　　数

MD＊256＋FNO (r)	word	モードおよびキー番号
	MD×256＋	FNO
BUFFER (r, w)	address	データバッファのポインタ

コ ー ル

```
         PEA       BUFFER
         MOVE. W   MODE, -(SP)
         DC. W     _FNCKEY
         ADDQ. L   #6, SP
           ⋮
BUFFER:  DS. B     32
```



リターン　なし

解　　説　モード MD の値によって、再定義可能なキーの読み出し／設定を行います。
この DOS コールは、CON デバイス（コンソール）でサポートされます。
MODE の上位 8 ビットにモード MD を、下位 8 ビットにキー番号 FNO を指定します。
モード MD には次の 2 種類があります。

MD ＝ 0……読み出し
MD ＝ 1……設定（同時にファンクションキーの表示も行いますが 32 行モードのときは表示しません）

キー番号FNOの意味は次の通りです。

FNO	キー名
0	すべてのキー
1～10	F1～F10
11～20	SHIFT+F1～F10
21	ROLL UP
22	ROLL DOWN
23	INS
24	DEL
25	UP(↑)
26	LEFT(←)
27	RIGHT(→)
28	DOWN(↓)
29	CLR
30	HELP
31	HOME
32	UNDO

BUFFERは読み出し／設定のためのデータバッファです。
バッファの大きさは、FNOの値によって異なります。

FNO ＝ 0……………712バイト (20×32バイト＋12×6バイト)
FNO ＝ 1～20………32バイト (31バイト＋$00)
FNO ＝ 21～32 ……6バイト (5バイト＋$00)

knjctrl

● knjctrl (MD, ??)　　　DOS コール番号：$FF22

機　能	かな漢字変換用ファンクションコールを呼び出します。
解　説	この DOS コールは、日本語 FP のファンクションコールで使用します。 詳しくは本書第 4 章をご覧ください。

conctrl

● conctrl (MD [, CODE] [, MESPTR] [, ATR] [, X, Y] [, N] [, MOD] [, YS, YL])　　DOS コール番号：$FF23

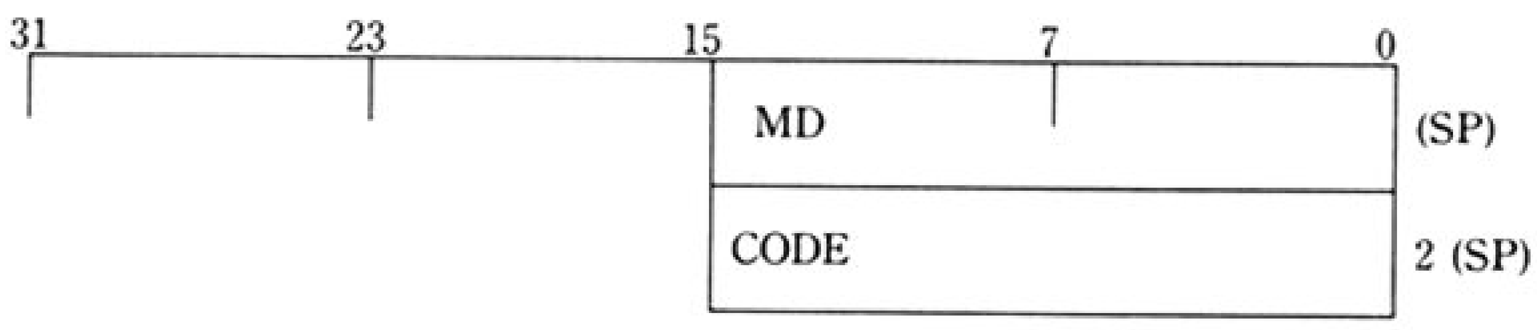
機　　能　コンソールに直接出力します（コンソールを制御）。

引　　数

MD (r)	word	コントロールコード（この値によって引数が異なります）
CODE (r)	word	1バイトデータ
MESPTR (r)	address	文字列データのポインタ
ATR (r)	word	文字属性
X (r)	word	カーソル位置指定
Y (r)	word	カーソル位置指定
N (r)	word	カーソル相対移動指定
MOD (r)	word	画面制御指定
YS (r)	word	スクロール範囲指定
YL (r)	word	スクロール範囲指定

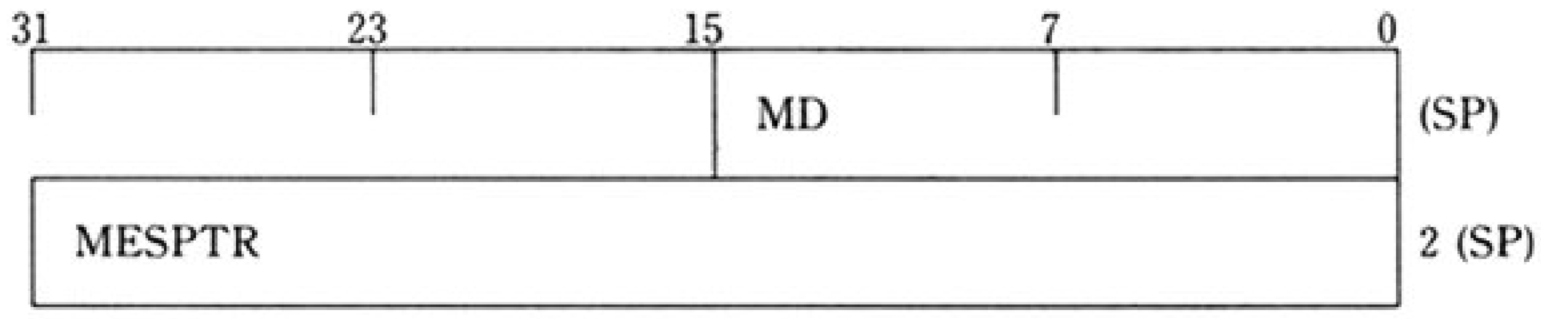
コ ー ル　<MD = 0>

```
        MOVE. W     CODE, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _CONCTRL
        ADDQ. L     #4, SP
```

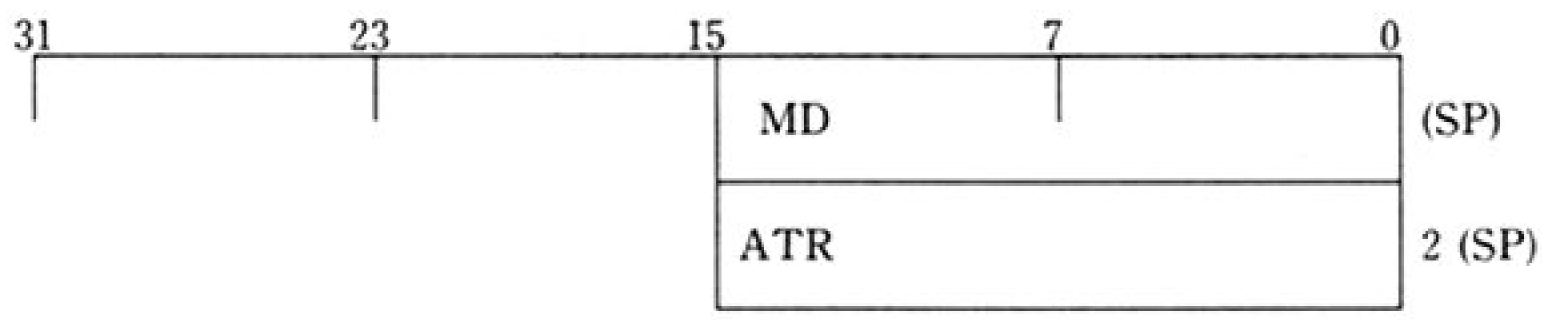


<MD = 1>

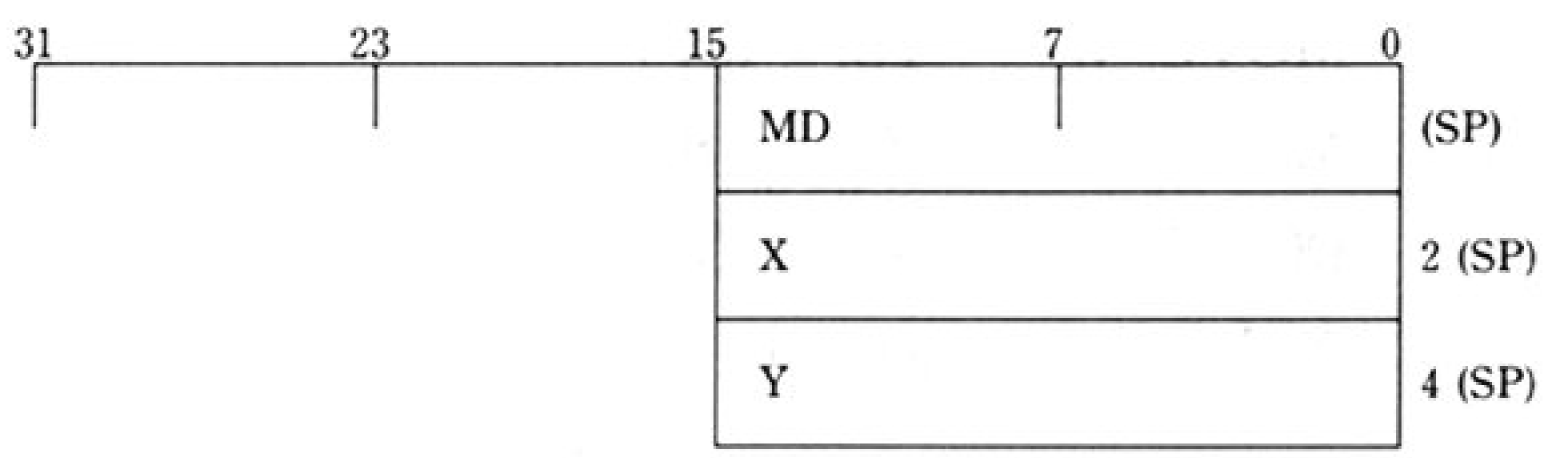
```
            PEA         MESPTR
            MOVE. W     MD, -(SP)
            DC. W       _CONCTRL
            ADDQ. L     #6, SP
              ⋮
MESPTR:     DC. B       'SHARP', 13, 10, 0
```



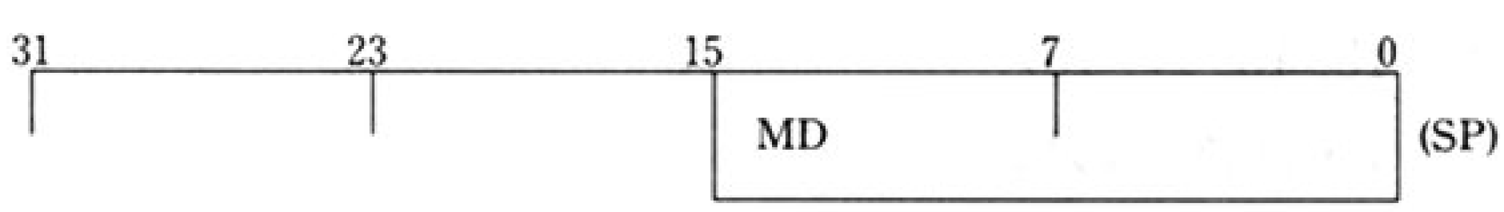

```
<MD = 2> MOVE. W    #ATR, -(SP)
         MOVE. W    MD, -(SP)
         DC. W      _CONCTRL
         ADDQ. L    #4, SP
```




```
<MD = 3>
         MOVE. W    Y, -(SP)
         MOVE. W    X, -(SP)
         MOVE. W    MD, -(SP)
         DC. W      _CONCTRL
         ADDQ. L    #6, SP
```

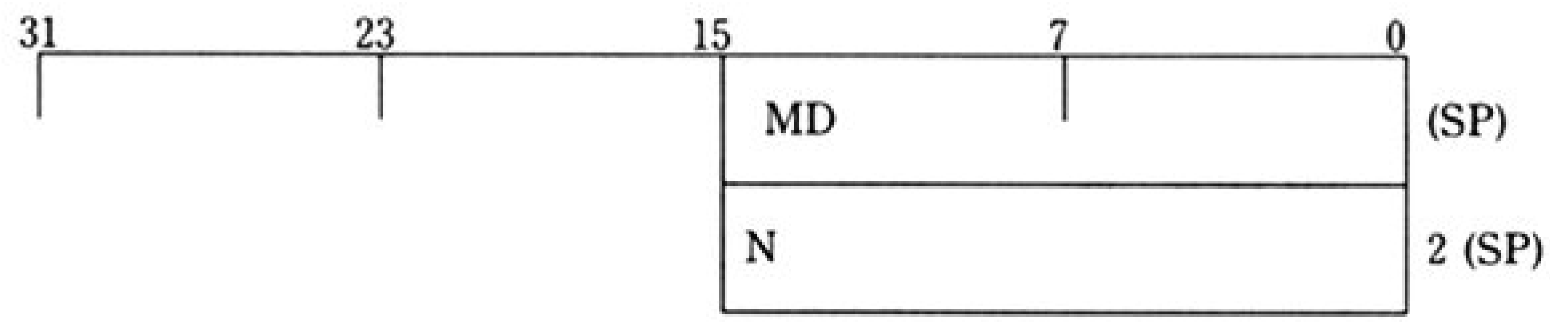



```
<MD = 4, 5, 17, 18>
         MOVE. W    MD, -(SP)
         DC. W      _CONCTRL
         ADDQ. L    #2, SP
```



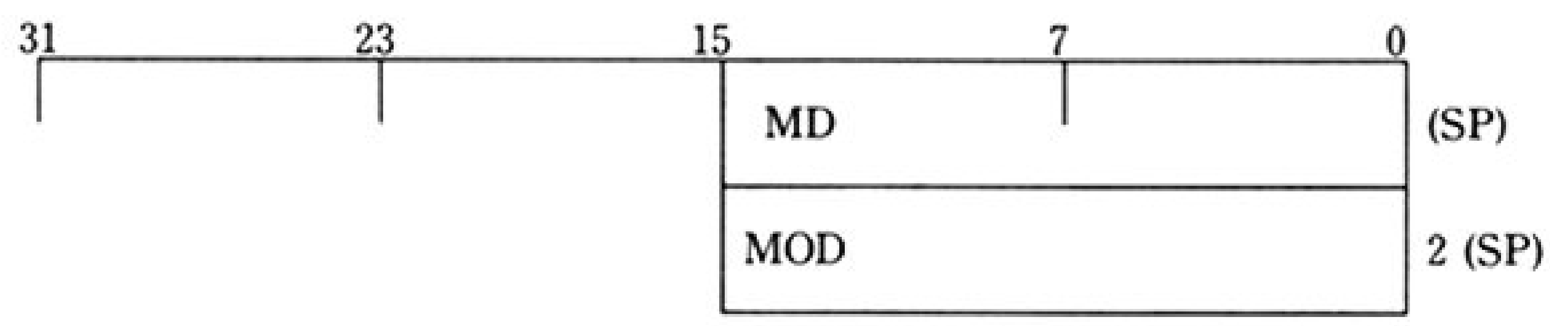

```
<MD = 6, 7, 8, 9, 12, 13>
         MOVE. W    N, -(SP)
```

```
MOVE. W     MD, -(SP)
DC. W       _CONCTRL
ADDQ. L     #4, SP
```



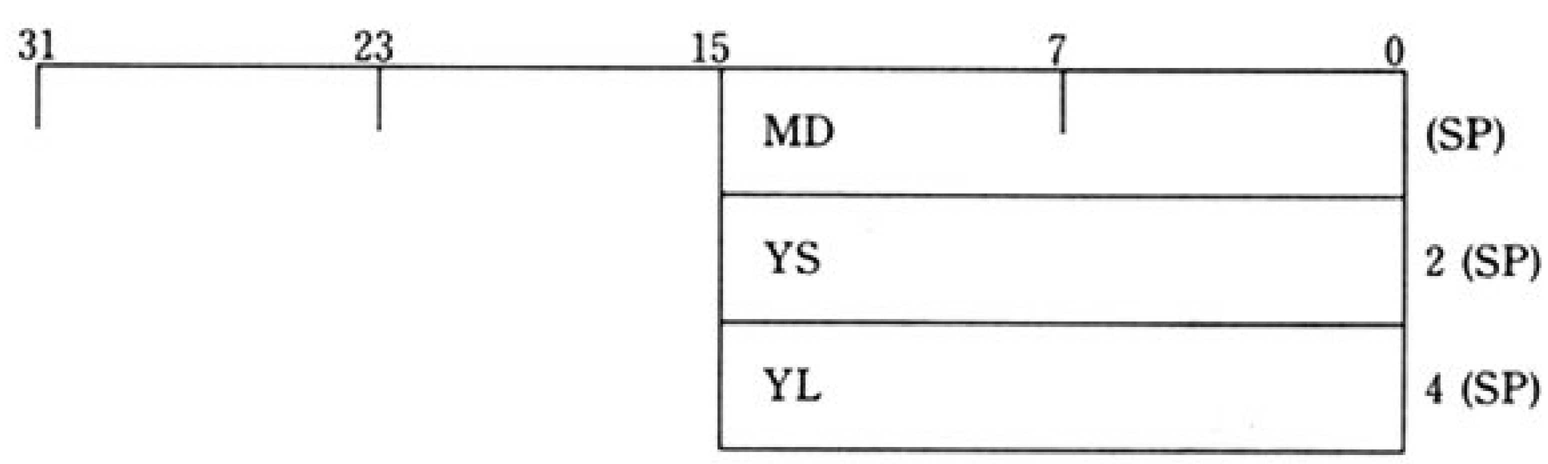
〈MD = 10, 11, 14, 16〉

```
MOVE. W     MOD, -(SP)
MOVE. W     MD, -(SP)
DC. W       _CONCTRL
ADDQ L      #4, SP
```



〈MD = 15〉

```
MOVE. W     YL, -(SP)
MOVE. W     YS, -(SP)
MOVE. W     MD, -(SP)
DC. W       _CONCTRL
ADDQ.L      #6, SP
```



リターン 〈MD = 0〜13, 15, 17, 18〉

D0. L　0（正常終了の場合）

〈MD = 14, 16〉

D0. L　前のモード

解　説　MDの値によって、コンソールから直接出力したり、コンソールを制御したりします。

この DOS コールは CON デバイスでサポートされます。

コントロールモード MD の内容は次の通りです。

MD = 0　　conctrl (0, CD)
　CDで指定した1バイトデータを表示します。

MD = 1　　conctrl (1, MESPTR)
　MESPTRで指定した文字列データを表示します。
　文字列データはヌル文字(0)で終了します。

MD = 2　　conctrl (2, ATR)
　ATRで指定した文字属性を設定します。

MD = 3　　conctrl (3, X, Y)
　X、Yで指定したカーソル位置に設定します。

MD = 4　　conctrl (4)
　1行下へカーソルを移動します。
　最下行ではスクロールアップします。

MD = 5　　conctrl (5)
　1行上へカーソルを移動します。
　先頭行ではスクロールダウンします。

MD = 6　　conctrl (6, N)
　N行上へカーソルを移動します。
　スクロールはしません。
　N = 0はN = 1と同じです。

MD = 7　　conctrl (7, N)
　N行下へカーソルを移動します。
　スクロールはしません。
　N = 0はN = 1と同じです。

MD = 8　　conctrl (8, N)
　N行右へカーソルを移動します。
　スクロールはしません。
　N = 0はN = 1と同じです。

MD = 9　　conctrl (9, N)
　N行左へカーソルを移動します。
　スクロールはしません。
　N = 0はN = 1と同じです。

MD = 10　　conctrl (10, MOD)

MODの値に従って、画面をクリアします。

モードMODの内容は次の通りです。

MOD = 0……カーソル位置から最終行右端まで。

MOD = 1……先頭行左端からカーソル位置まで。

MOD = 2……画面全体。

カーソルはホームに移動します。

MD = 11　　conctrl (11, MOD)

モードMODの値に従って、現在行をクリアします。

モードMODの内容は次の通りです。

MOD = 0……カーソル位置から行の右端まで。

MOD = 1……行の左端からカーソル位置まで。

MOD = 2……カーソルのある行1行。

MD = 12　　conctrl (12, N)

カーソル行にN行挿入します。

N = 0はN = 1と同じです。

MD = 13　　conctrl (13, N)

カーソルからN行削除します。

N = 0はN = 1と同じです。

MD = 14　　conctrl (14, MOD)

MODの値に従って、ファンクションキー行のモードを指定します。

スクロール範囲はリセットされます。

モードMODの内容は次の通りです。

MOD = 0………ファンクションキーを表示（スクロール範囲は0から31行分）

MOD = 1………シフトファンクションキーを表示（スクロール範囲は0から31行分）

MOD = 2………なにも表示しません（スクロール範囲は0から31行分）

MOD = 3………ファンクションキー行を他の行と区別しません（スクロール範囲は0から32行分）

MOD = −1……現在のモードをD0に返します。

リターンのD0. Lレジスタには前のモード（0〜3）を返します。

MD = 15　　conctrl (15, YS, YL)

YS、YLでスクロール範囲を設定します。

スクロール後、カーソルはホームポジションに移動します。

YS+YLの値は、ファンクションキー行のモードが3（MD = 14、MOD = 3)のときは32まで、それ以外は31までです。

絶対座標の(0, YS)が論理座標の(0, 0)となります。

MD ＝ 16　　conctrl (16, MOD)

MODで指定する画面サイズにします。

モードMODの内容は次の通りです。

MOD ＝ 0………高解像度768×512、グラフィックなし

MOD ＝ 1………高解像度768×512、グラフィック16色

MOD ＝ 2………高解像度512×512、グラフィックなし

MOD ＝ 3………高解像度512×512、グラフィック16色

MOD ＝ 4………高解像度512×512、グラフィック256色

MOD ＝ 5………高解像度512×512、グラフィック65536色

MOD ＝ −1……前のモードを調べます

リターンのD0. Lレジスタには前のモード（0〜5）を返します。

MD ＝ 17　　conctrl (17)

カーソルを表示するモードにします。

MD ＝ 18　　conctrl (18)

カーソルを表示しないモードにします。

上記説明中リターンのD0. Lレジスタについて説明していないものについては、コールが成功した場合に0を返します。

keyctrl

● keyctrl (MD [, GROUP] [, INSMODE])　　　DOS コール番号：$FF24

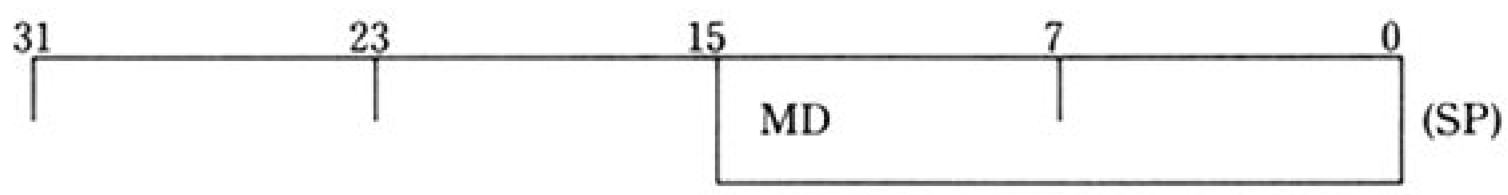
機　　能　コンソールから直接入力します。

引　　数

MD (r)	word	コントロールコード（この値によって引数が異なります）
GROUP (r)	word	指定キーコードグループ
INSMODE (r)	word	INS キーのモード

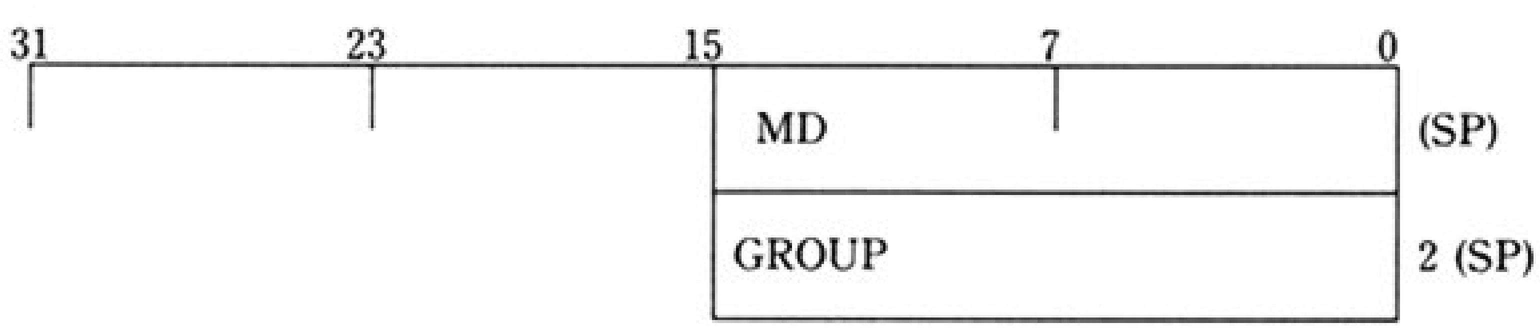
コ　ー　ル　〈MD = 0, 1, 2〉

```
        MOVE W      MD, -(SP)
        DC. W       _KEYCTRL
        ADDQ. L     #2, SP
```



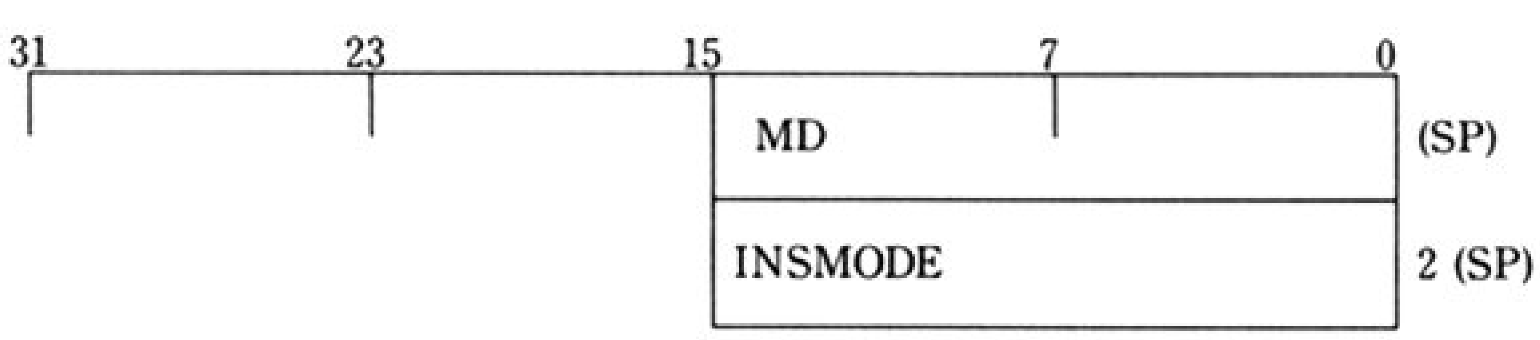
〈MD = 3〉

```
        MOVE. W     GROUP, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _KEYCTRL
        ADDQ. L     #4, SP
```



〈MD = 4〉

```
        MOVE. W     INSMODE, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _KEYCTRL
        ADDQ. L     #4, SP
```



リターン　D0. L　〈MD ＝ 0, 1〉

D0. L　入力されたキーコード

〈MD ＝ 2, 3〉

D0. L　キーの状態

〈MD ＝ 4〉

なし

解　　説　MDの値によって、直接コンソールから入力したり、コンソールの状態を調べたりします。

このDOSコールはCONデバイスでサポートされます。

コントロールコードMDの内容は次の通りです。

MD ＝ 0　keyctrl (0)

キーを入力します。

入力したキーコードをD0. Lレジスタに返します。

MD ＝ 1　keyctrl (1)

キーコードを先読みします。

キーコードをD0. Lレジスタに返します。

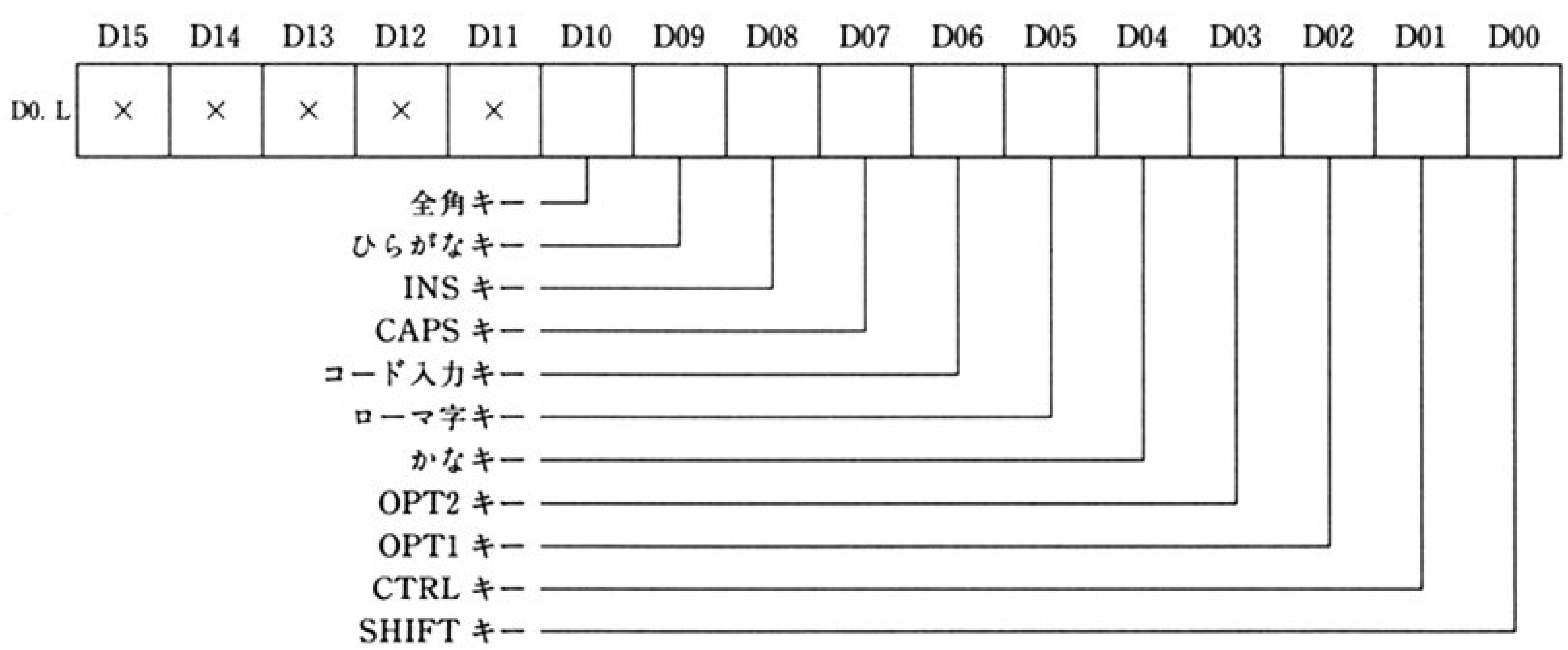
D0. L ＝ 0なら、入力なしだったことを示します。

MD ＝ 2　keyctrl (2)

シフトキーの状態を調べます。

D0. Lレジスタの各ビットは、次のようにシフトキーに対応しています。

ビット ＝ 1なら対応するキーが押されています。



MD = 3　　keyctrl (3, GROUP)

GROUPで指定したキーコードグループのキー状態を、D0. Lレジスタに返します。

キーコードグループとキー状態との関係は、本書第3章IOCSコールB_BITSNS ($04)を参照して下さい。

MD = 4　　keyctrl (4, INSMODE)

[INS]キーをON／OFFします。

INSキーモードINSMODEの値は次の通りです。

INSMODE = $FF…[INS]キー　ON

INSMODE = 0　…[INS]キー　OFF

intvcs

● intvcs (INTNO, JOBADR)　　DOS コール番号：$FF25

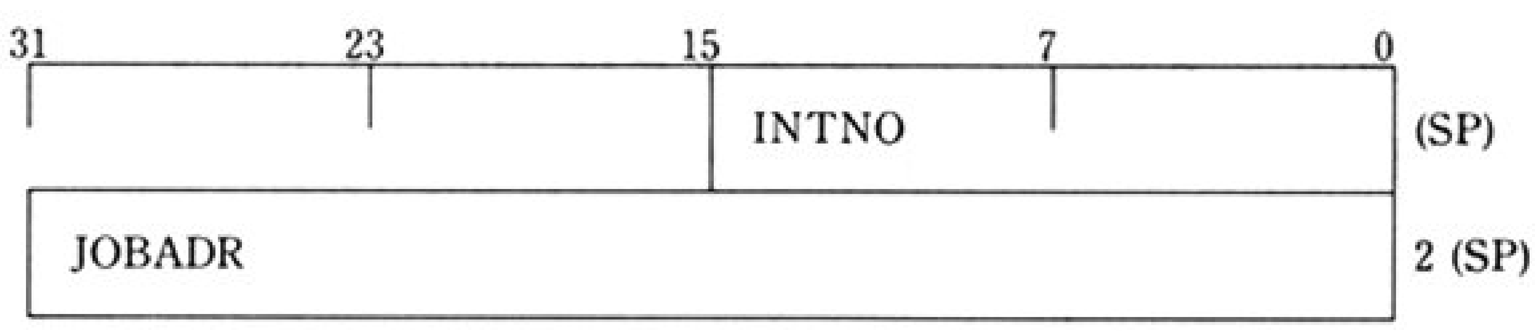
機　能　INTNOで指定するベクタにアドレスを設定します。

引　数

INTNO (w)	word	ベクタ
JOBADR (r)	address	INTNOで指定するベクタの処理アドレス

コール

```
        PEA      JOBADR
        MOVE. W  INTNO, -(SP)
        DC. W    _INTVCS
        ADDQ. L  #6, SP
          ⋮
JOBADR:
          ⋮
        RTE      * 0~$1FF のベクタの場合
        (RTS)    * Human68k の DOS コールの場合
```



リターン　D0. L　　前の INTNO ベクタの処理アドレス

解　説　INTNOで指定するベクタに、JOBADRをセットします。
ベクタINTNOの内容は次の通りです。

INTNO ＝ 0~$FF …………割り込み処理
INTNO ＝ $100~$1FF ……IOCS コール
INTNO ＝ $FF00~$FFFF…DOS コール

リターン値（D0. L）として、前のベクタの処理アドレスを返します。

pspset

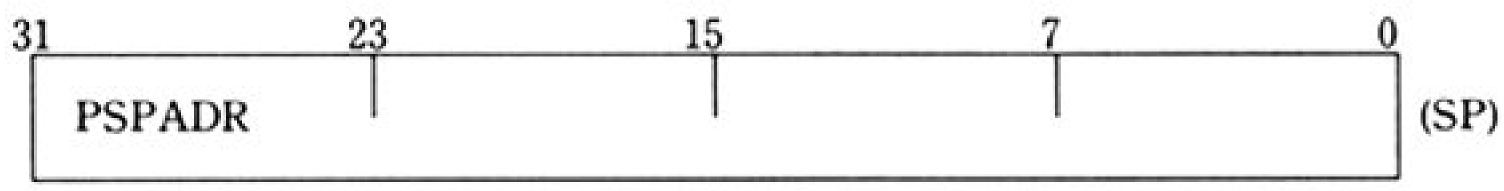
● pspset (PSPADR)　　DOS コール番号：$FF26

機　　能　バッファにプロセス管理情報を作成します。

引　　数　PSPADR (w)　address　　プロセス管理情報のポインタ

コ ー ル

```
        PEA       PSPADR
        DC. W     _PSPSET
        ADDQ. L   #4, SP
```



リターン　なし

解　　説　PSPADR で指定したバッファに、プログラム終了時のさまざまな情報を作成します。

プロセス管理ポインタのバッファの大きさは 240 バイトです。

プロセス管理ポインタ PSPADR は、メモリ管理ポインタ+$10 のアドレスでなくてはなりません。

これには、malloc ($FF48) のリターンアドレスを指定してください。

gettim2

● gettim2 ( )　　　DOS コール番号：$FF27

機　　能　現在の時刻を得ます。

引　　数　なし

コ ー ル　DC. W　　_GETTIM2

リターン　D0. L　　現在の時刻

解　　説　現在の時刻を求めます。
リターンの D0. L レジスタに、現在の時刻を返します。
時刻データのフォーマットは次の通りです。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0. L	0	0	0	0	0	0	0	0	0	0	0	時(0～23 時)				

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	分(0～59 分)						0	0	秒(0～59 秒)					

gettime ($FF2C) との違いは、秒が 1 秒単位なので、リターン値（D0. L）がロングワード（4 バイト）になっていることです。

settim2

● settim2 (TIME)　　DOS コール番号：$FF28

機能　時刻を設定します。

引数　TIME (r)　long　時刻

コール

```
MOVE. L     TIME, -(SP)
DC. W       _SETTIM2
ADDQ. L     #4, SP
```

31–0	
TIME	(SP)

リターン　D0. L　エラーコード

解説　時刻を TIME の値に設定します。
時刻データ TIME のフォーマットは次の通りです。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
TIME	0	0	0	0	0	0	0	0	0	0	0	時(0～23 時)				

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	分(0～59 分)						0	0	秒(0～59 秒)					

settime ($FF2D)との違いは、秒を 1 秒単位にしているため、TIME がロングワード（4バイト）になっていることです。

namests

● namests (FILE, BUFFER)　　　DOS コール番号：$FF29

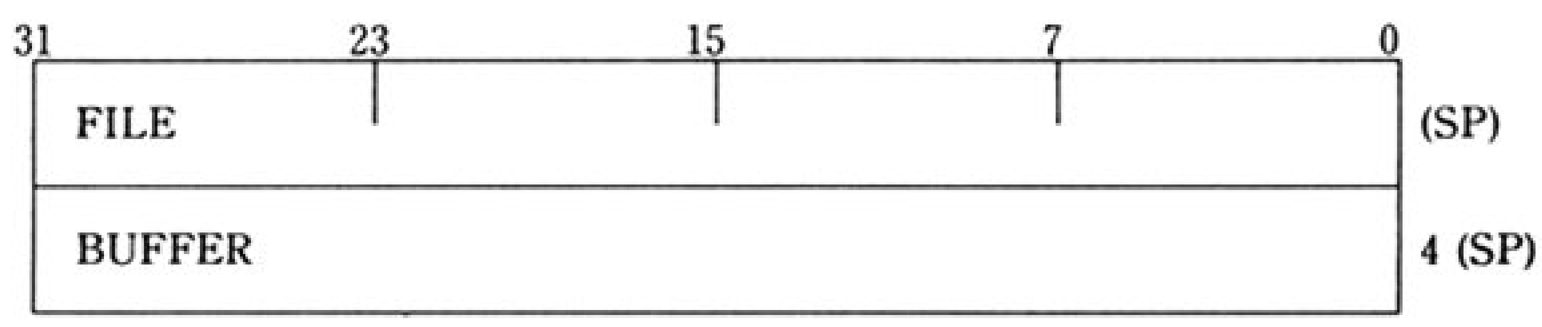
機　　能　ファイル名をバッファに展開します。

引　　数

FILE (r)　　address　　文字列（ファイル名）のポインタ
BUFFER(w) address　　バッファのポインタ

コ ー ル

```
         PEA      BUFFER
         PEA      FILE
         DC. W    _NAMESTS
         ADDQ. L  #8, SP
           ⋮
FILE:    DC. B    'B:¥TEST¥ABCDEFGHIJKLMN. S', 0
BUFFER:  DS. B    88
```



リターン　D0. L　　エラーコード

解　　説　FILEで指定したファイル名を、BUFFERで指定したバッファに展開します。バッファBUFFERは88バイトで、フォーマットは次の通りです。

```
BUFFER:
NAMWLD: DS. B   1    *0ならワイルドカードなし、$FFならフ
                      ァイル指定なし
NAMDRV: DS. B   1    *ドライブ番号（A ＝ 0、B ＝ 1）
NAMPTH: DS. B   65   *パス（ルート記号'¥'と、あればサブディ
                      レクトリ名＋'¥'）
NAMNM1: DS. B   8    *ファイル名(先頭8文字まで)
NAMEXT: DS. B   3    *ファイル名の拡張子
NAMNM2: DS. B   10   *ファイル名（残りの10文字）
                      計88バイト
```

ドライブ番号（NAMDRV）は次の通りです。

NAMDRV ＝ 0……Aドライブ
NAMDRV ＝ 1……Bドライブ

NAMDRV = 2……Cドライブ

⋮　⋮

コールのようにFILEを設定していれば、バッファに次のように展開します。

```
BUFFER: DC. B    0       *ワイルドカード使用せず、ファイル名指定
                          あり
        DC. B    1       *Bドライブ
        DC. B    '¥TEST¥', 0
        DS. B    65-7    *'¥TEST¥'+0で7バイトしか使用しなか
                          った
        DC. B    'ABCDEFGH'
        DC. B    'S  '
        DC. B    'IJKLMN', 0, 0, 0, 0
```

getdate

● getdate ( )　　DOS コール番号：$FF2A

機　　能　日付を得ます。

引　　数　なし

コ ー ル　DC. W　_GETDATE

リターン　D0. L　現在の日付

解　　説　現在の日付を求めます。
リターンのD0. Lレジスタに、現在の日付を返します。
日付データのフォーマットは次の通りです。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0. L	0	0	0	0	0	0	0	0	0	0	0	0	0	曜日(0～6)		

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
年－1980(0～99)							月(1～12月)				日(1～31日)				

このうち、年は西暦から1980を引いた値が、0～99年まで格納されています。
曜日は、0～6が日～土曜日を表します。

setdate

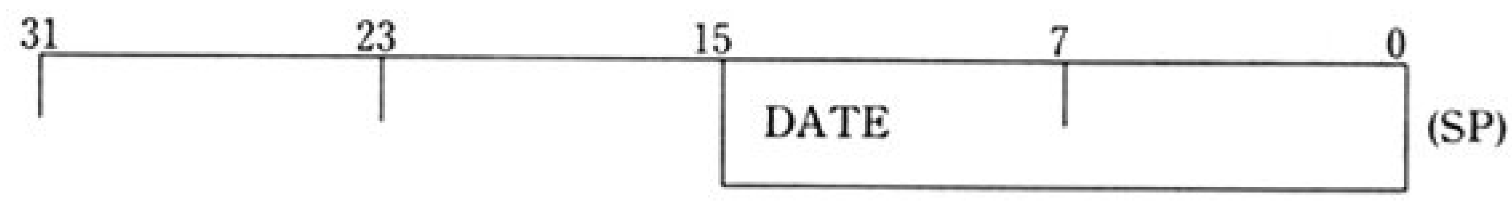
● setdate (DATE)　　DOS コール番号：$FF2B

機　　能　日付を設定します。

引　　数　DATE (r)　word　日付

コ ー ル

```
MOVE. W   DATE, -(SP)
DC. W     _SETDATE
ADDQ. L   #2, SP
```



リターン　D0. L　エラーコード

解　　説　DATE の値を日付に設定します。

日付データフォーマットは次の通りです。

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
DATE	年−1980 (0〜99)							月 (1〜12 月)				日 (1〜31 日)				

このうち、年は西暦から 1980 を引いた値を 0〜99 年まで格納してください。

gettime

● gettime ( )　　DOS コール番号：$FF2C

機　　能　時刻を得ます。

引　　数　なし

コ ー ル　DC. W　　_GETTIME

リターン　D0. W　　現在の時刻

解　　説　現在の時刻を求めます。
リターンの D0. W レジスタに、現在の時刻を返します。
時刻データのフォーマットは次の通りです。

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
D0. W	時(0～23 時)					分(0～59 分)						秒／2(0～29)				

このうち、秒は 2 秒単位で表し、実際の秒を 2 で割った値になります。
したがって、値は 0～29 までです。
gettim2 ($FF27)との違いは、秒の単位が 2 秒のため、リターン値（D0. W）がワード（2 バイト）になっていることです。

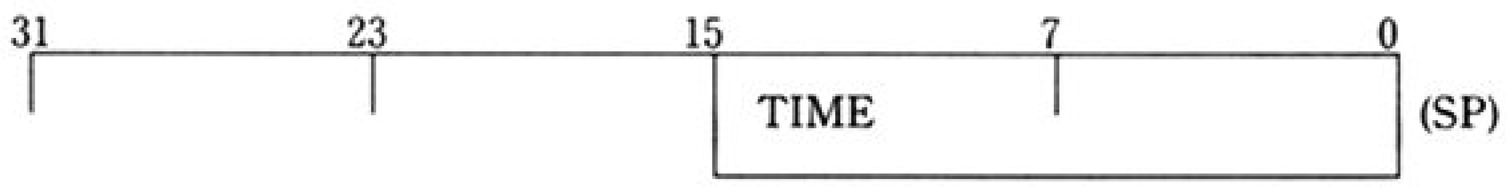
settime

● settime (TIME)　　DOS コール番号：$FF2D

機能	時刻を設定します。
引数	TIME (r)　word　時刻

コール

```
MOVE. W   TIME, -(SP)
DC. W     _SETTIME
ADDQ. L   #2, SP
```



リターン　D0. L　エラーコード

解説　時刻を TIME の値に設定します。
時刻データ TIME のフォーマットは次の通りです。

	D15–D11	D10–D05	D04–D00
TIME	時(0～23 時)	分(0～59 分)	秒／2(0～29)

このうち、秒は 2 秒単位で表し、実際の秒を 2 で割った値になります。
したがって、設定する値は 0～29 までとなります。
settim2 ($FF28)との違いは、秒を 2 秒単位にしているため、TIME がワード (2 バイト) になっていることです。

verify

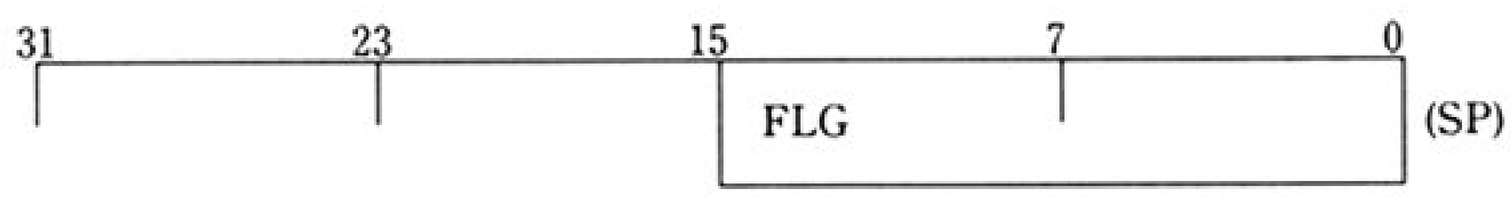
● verify (FLG)　　DOS コール番号：$FF2E

機　能　ベリファイフラグを設定します。

引　数　FLG (r)　word　　ベリファイフラグ

コール

```
MOVE. W     FLG, −(SP)
DC. W       _VERIFY
ADDQ. L     #2, SP
```



リターン　なし

解　説　ディスクなどの書き込み時のベリファイフラグを設定します。

ベリファイフラグ FLG の内容は次の通りです。

FLG ＝ 0……ベリファイしない

FLG ＝ 1……ベリファイする

dup0

● dup0 (FILENO, NEWNO)　　　DOS コール番号：$FF2F

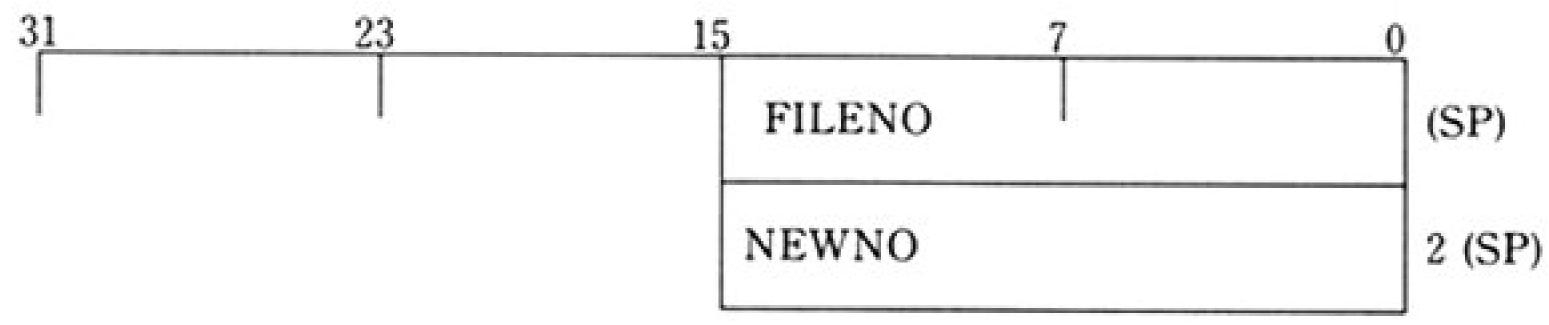
機　能	ファイルハンドルを強制複写します。

引　数

FILENO (r)	word	現在のファイルハンドル
NEWNO (w)	word	複写先のファイルハンドル

コール

```
MOVE. W     NEWNO, -(SP)
MOVE. W     FILENO, -(SP)
DC. W       _DUP0
ADDQ. L     #4, SP
```



リターン

D0. L　NEWNO の前の値

　D0. L　<　0……エラーコード

解　説

現在オープンしているファイルのファイルハンドル FILENO を、2つ目のファイルハンドル NEWNO に強制複写します。

これにより1つのファイルに、2つのファイルハンドルが与えられ、同一ファイルの操作を複数のファイルハンドルで行うことができるようになります。

CTTY 用のファイルハンドルは、dup ($FF45)／dup2 ($FF46) により、変更できます。

クローズすると、この番号に戻ります。

指定できる NEWNO は 0～4 までです。

それ以上は、dup2 ($FF46) で指定してください。

リターンの D0. L レジスタには、NEWNO の前の値を返します。

また、負の数ならエラーコードです。

dup0 (3, 2) ; dup0 (3, 1) ; dup0 (3, 0)　……標準入出力、標準エラー出力を AUX にします。

dup0 (2, 2) ; dup0 (1, 1) ; dup0 (0, 0)　……もとに戻します。

vernum

● vernum ( )　　DOS コール番号：$FF30

機　能　OS のバージョン番号を得ます。

引　数　なし

コール　DC. W　_VERNUM

リターン　D0. L　バージョン番号

解　説　Human68k のバージョン番号を求めます。
リターンの D0. L レジスタには、バージョン番号を返します。
バージョン番号データのフォーマットは次の通りです。

D0. L

D31 — D24	D23 — D16
'6' ($36)	'8' ($38)

D15 — D08	D07 — D00
バージョン番号の整数部	バージョン番号の小数部

keeppr

● keeppr (PRGLEN, CODE)　　DOS コール番号：$FF31

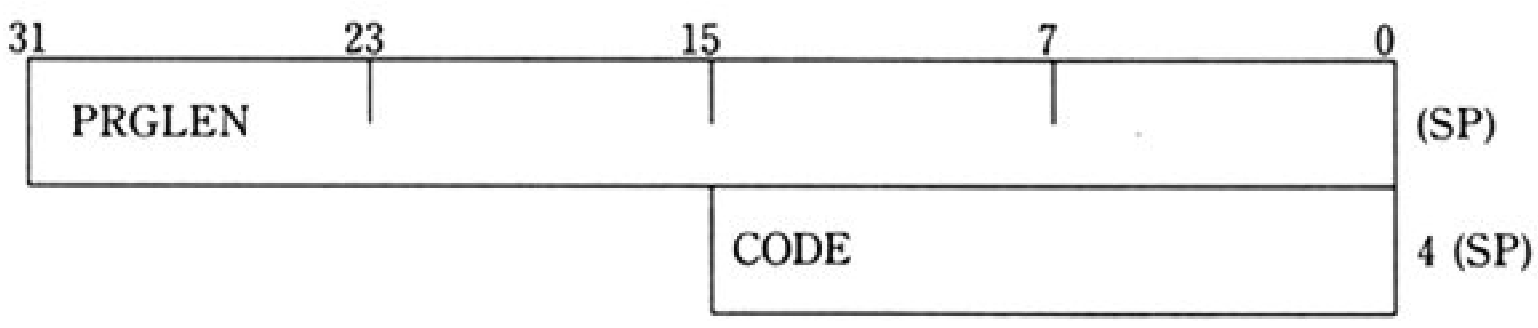
機　能　常駐終了します。

引　数

PRGLEN (r)	long	常駐バイト数
CODE (r)	word	終了コード

コール

```
MOVE. W   CODE, -(SP)
MOVE. L   PRGLEN, -(SP)
DC. W     _KEEPPR
ADDQ. L   #6, SP
```



リターン　なし

解　説　PRGLEN バイトを残して、プロセスを終了します（PSP 部分を含みません）。
CODE は終了コードです。
現在オープンしているファイルハンドルはクローズします。
子プロセスがオープンしたファイルハンドルもクローズします。

getdpb

● getdpb (DRIVE, DPBPTR)　　　DOS コール番号：$FF32

機　能　指定したドライブ番号のドライブパラメータブロックを複写します。

引　数

DRIVE (r)	word	ドライブ番号
DPBPTR (w)	address	ドライブパラメータブロックのポインタ

コール

```
            PEA        DPBPTR
            MOVE. W    DRIVE, -(SP)
            DC. W      _GETDPB
            ADDQ. L    #6, SP
              ⋮
DPBPTR:     DS. B      94
```

31–16	15–0	
	DRIVE	(SP)
DPBPTR		2 (SP)

リターン　D0. L

D0. L　<　0……エラーコード

解　説　DRIVEで指定したドライブ番号のドライブパラメータブロックを、バッファに複写します。

ドライブ番号DRIVEは次の通りです。

DRIVE　=　0　……カレントドライブ
DRIVE　=　1　……Aドライブ
DRIVE　=　2　……Bドライブ
DRIVE　=　3　……Cドライブ
　　⋮　　　　⋮

ドライブパラメータブロックDPBPTRは94バイトでフォーマットは次の通りです。

```
DPBPTR:  DS. B   1   *ドライブ番号、0 = A、1 = B
         DS. B   1   *デバイスドライバで使うユニット番号
         DS. W   1   *1セクタあたりのバイト数
         DS. B   1   *1クラスタあたりのセクタ数-1
         DS. B   1   *先頭クラスタのセクタ番号
```

```
DS.W    1     * FATの先頭セクタ番号
DS.B    1     * FAT領域の個数
DS.B    1     * FATの占めるセクタ数(複写分を除く)
DS.W    1     *ルートディレクトリの個数
DS.W    1     *データ部の先頭セクタ番号
DS.W    1     *総クラスタ数+1
DS.W    1     *ルートディレクトリの先頭セクタ番号
DS.L    1     *デバイスドライバへのポインタ
DS.B    1     *メディアバイト
DS.B    1     * DPB使用フラグ(−1でアクセスなし)
DS.L    1     *次のDPBのポインタ
DS.W    1     *カレントディレクトリのクラスタ番号(0は
               ルートを表す)
DS.B    64    *カレントディレクトリの文字バッファ
              *合計94バイト
```

breakck

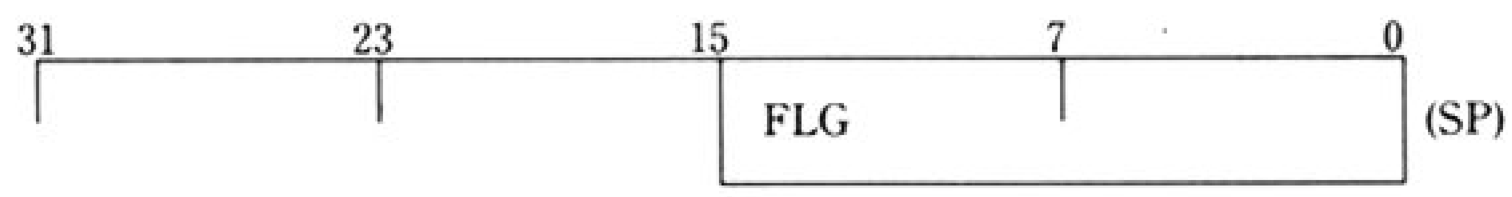
● breakck (FLG)　　DOS コール番号：$FF33

機　能　ブレークチェックを設定します。

引　数　FLG (r)　word　フラグ

コール

```
MOVE. W   FLG, -(SP)
DC. W     _BREAKCK
ADDQ. L   #2, SP
```



リターン　D0. L　　設定状況（FLG ＝ −1 のとき）

D0. L ＝ 0 ……指定の DOS コールのみチェック
D0. L ＝ 1 ……すべての DOS コールでチェック
D0. L ＝ 2 ……すべての DOS コールでチェックしない

解　説　ブレークチェックを設定します。

次に示すように、FLG の値によってどのようにチェックするかを選択します。

FLG ＝ 0………指定の DOS コールだけチェックする
FLG ＝ 1………すべての DOS コールでチェックする
FLG ＝ 2………すべての DOS コールでチェックしない
FLG ＝ −1……設定状況を調べる

「指定の DOS コール」とは、「ブレークチェックをします」と書かれた以下の DOS コールのことです。

getchar ($FF01)	print ($FF09)
putchar ($FF02)	gets ($FF0A)
cominp ($FF03)	keysns ($FF0B)
comout ($FF04)	kflush ($FF0C)
prnout ($FF05)	fputc ($FF0D)
getc ($FF08)	fputs ($FF0E)

drvxchg

● drvxchg (OLD, NEW)　　DOS コール番号：$FF34

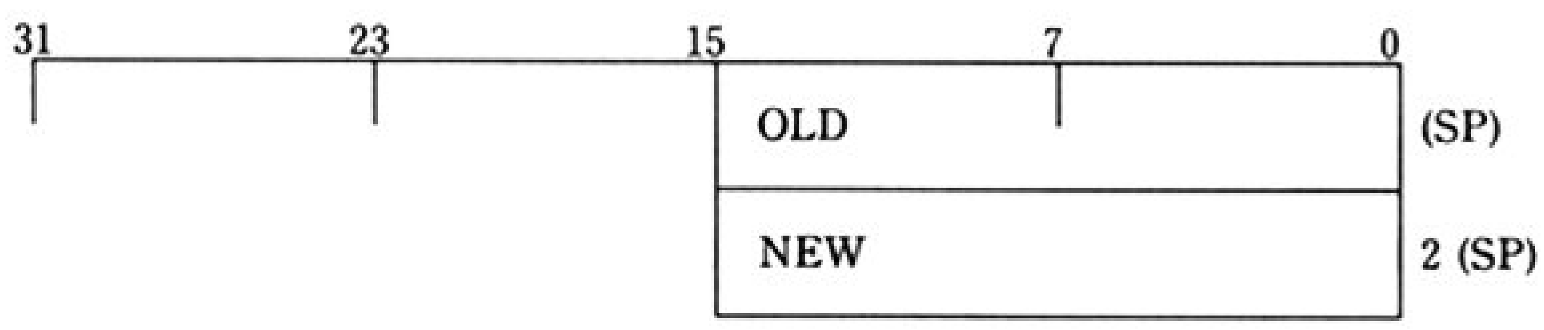
機　能　ドライブを入れ替えます。

引　数

OLD (r)	word	ドライブ番号
NEW (r)	word	ドライブ番号

コール

```
MOVE. W   NEW, −(SP)
MOVE. W   OLD, −(SP)
DC. W     _DRVXCHG
ADDQ. L   #4, SP
```



リターン　D0. L　カレントドライブの新しいドライブ番号 (OLD ＝ −1, NEW ＝ −1 の時)

D0. L ＝ 0 ……A ドライブ
D0. L ＝ 1 ……B ドライブ
D0. L ＝ 2 ……C ドライブ
⋮　　　⋮

解　説　OLD で指定したドライブと、NEW で指定したドライブを入れ替えます。
OLD、NEW のドライブ番号には、次の番号が指定できます。

0 ……カレントドライブ
1 ……A ドライブ
2 ……B ドライブ

また、OLD, NEW ともに−1 と指定すると、すべてのドライブ変更を元に戻します。
そのとき、リターン値 (D0. L) としてカレントドライブの新しいドライブ番号が返ります。

intvcg

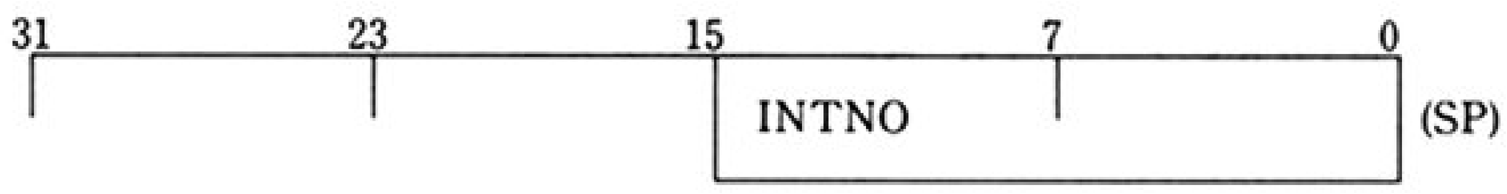
● intvcg (INTNO)　　DOS コール番号：$FF35

機　能　INTNOで指定するベクタの処理アドレスを得ます。

引　数　INTNO (r)　word　ベクタ

コ ー ル

```
MOVE. W   INTNO, -(SP)
DC. W     _INTVCG
ADDQ. L   #2, SP
```



リターン　D0. L　　INTNOベクタの処理アドレス

解　説　INTNOで指定するベクタの処理アドレスを求めます。
ベクタ INTNO の内容は次の通りです。

INTNO ＝ $00～$FF …………割り込み処理
INTNO ＝ $100～$1FF ………IOCS コール
INTNO ＝ $FF00～$FFFF …DOS コール

dskfre

● dskfre (DRIVE, BUFFER)　　DOS コール番号：$FF36

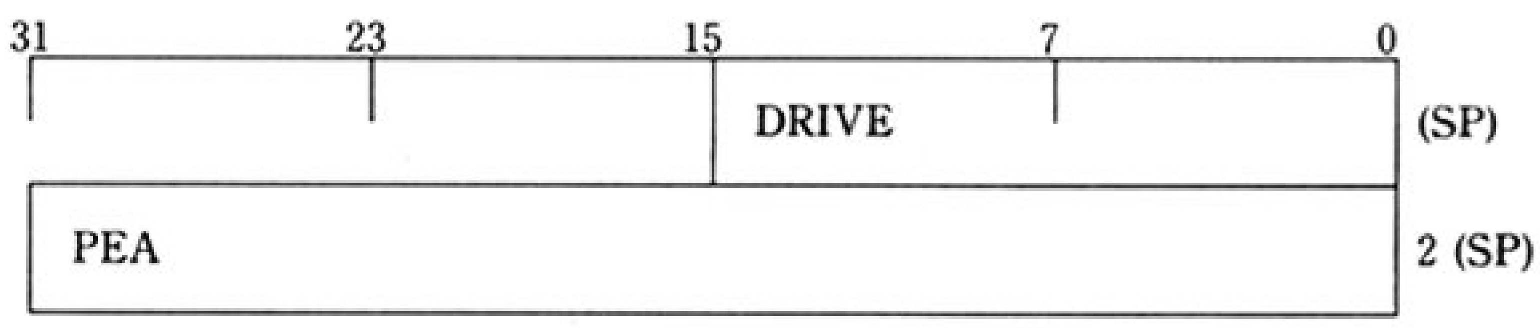
機　能	ディスクの残り容量を得ます。

引　数

DRIVE (r)	word	ドライブ番号
BUFFER (w)	address	バッファのポインタ

コール

```
        PEA       BUFFER
        MOVE. W   DRIVE, -(SP)
        DC. W     _DSKFRE
        ADDQ. L   #6, SP
          ⋮
BUFFER: DS. W     4
```



リターン　D0. L　　使用可能なバイト数（最大 2 ギガバイト）

　D0. L　＜　0……エラーコード

解　説　DRIVE で指定したドライブのディスクの残り容量を調べ、結果を BUFFER で指定したバッファに書き込みます。

バッファ BUFFER は 8 バイトでフォーマットは次の通りです。

```
BUFFER:DS. W   1   ＊使用可能なクラスタ数
       DS. W   1   ＊総クラスタ数
       DS. W   1   ＊1 クラスタあたりのセクタ数
       DS. W   1   ＊1 セクタあたりのバイト数
```

nameck

● nameck (FILE, BUFFER)　　　DOS コール番号：$FF37

機　能　ファイル名をバッファに展開します。

引　数

FILE (r)	address	文字列（ファイル名）のポインタ
BUFFER (w)	address	バッファのポインタ

コール

```
         PEA      BUFFER
         PEA      FILE
         DC. W    _NAMECK
         ADDQ. L  #8, SP
           ⋮
FILE:    DC. B    'B:¥TEST¥ABCDEFGH. S', 0
BUFFER:  DS. B    91
```

31　　　23　　　15　　　7　　　0	
FILE	(SP)
BUFFER	4 (SP)

リターン　D0. L

D0. L ＝ $FF ………ファイル指定なし
D0. L ＝ 0 …………ワイルドカード指定なし
D0. L ＜ 0 …………エラーコード
D0. L ＝ 上記以外 …ワイルドカード指定あり

解　説　FILE で指定したファイル名を、BUFFER で指定したバッファに展開します。
バッファ BUFFER は 91 バイトで、フォーマットは次の通りです。

```
BUFFER:DS. B   1+1     *ドライブ名(ドライブ名+'):'
       DS. B   64+1    *パス名(ルートなら'¥'+0 もしサブディレク
                        トリ名があれば'¥'+サブディレクトリ名+
                        '¥'+0)
       DS. B   18+1    *ファイル名(18 文字まで+0)
       DS. B   1+3+1   *拡張子('. '+拡張子+0)
                       *合計 91 バイト
```

コールのように使用すると、バッファ BUFFER には次のように展開します(エラーの場合は意味がありません)。

```
BUFFER:  DC. B   'B:'
         DC. B   '¥TEST¥', 0
         DS. B   65-7         *エリアの残り
         DC. B   'ABCDEFGHIJKLMN', 0
         DS. B   19-15        *エリアの残り
         DC. B   '. S,' 0
         DS. B   5-3          *エリアの残り
```

mkdir

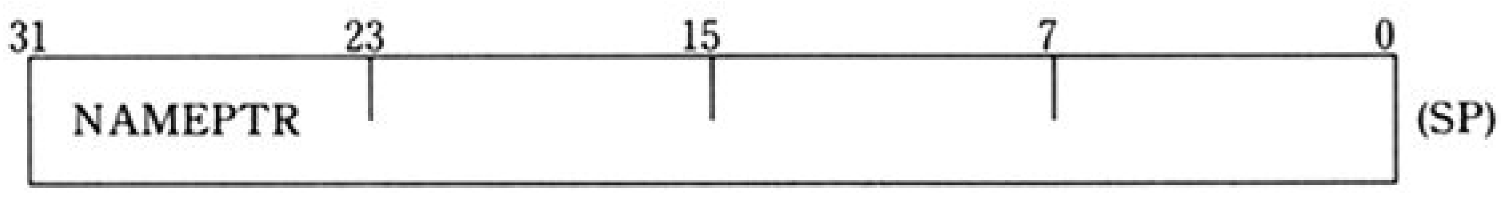
● mkdir (NAMEPTR)　　DOS コール番号：$FF39

機　　能　サブディレクトリを作成します。

引　　数　NAMEPTR (r)　address　　文字列（ディレクトリ名）へのポインタ

コ ー ル

```
            PEA       NAMEPTR
            DC. W     _MKDIR
            ADDQ. L   #4, SP
              ⋮
NAMEPTR:    DC. B     'A: ¥DOC', 0
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　　説　NAMEPTRで指定したディレクトリ名で、サブディレクトリを作成します。

リターンのD0. Lレジスタに通常は0が返りますが、特殊デバイスドライバが対象の時は0以外の正の数が返る場合があります。

負の数の場合は、エラーコードを表します。

rmdir

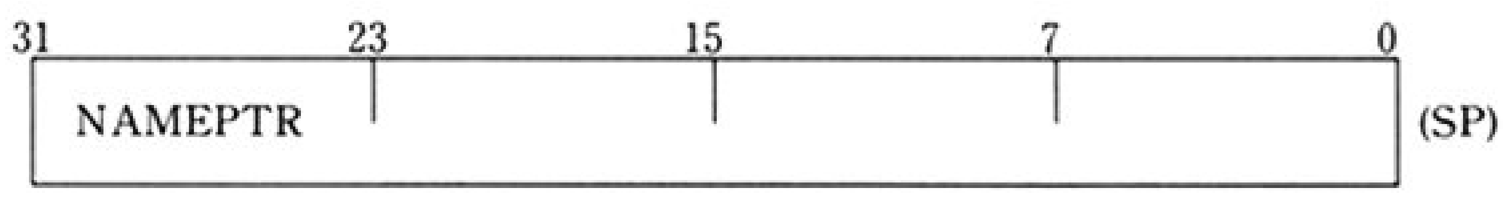
● rmdir (NAMEPTR)　　　DOS コール番号：$FF3A

機　　能　サブディレクトリを削除します。

引　　数　NAMEPTR (r)　address　　文字列（ディレクトリ名）へのポインタ

コ ー ル

```
            PEA       NAMEPTR
            DC. W     _RMDIR
            ADDQ. L   #4, SP
              ⋮
NAMEPTR:    DC. B     'A : ¥DOC', 0
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　　説　NAMEPTR で指定したディレクトリ名の、サブディレクトリを削除します。
指定するディレクトリは、空でなければなりません。
また、カレントディレクトリ、ルートディレクトリは削除できません。
リターンの D0. L レジスタに通常は 0 が返りますが、特殊デバイスドライバが対象の時は 0 以外の正の数が返る場合があります。
負の数の場合は、エラーコードを表します。

chdir

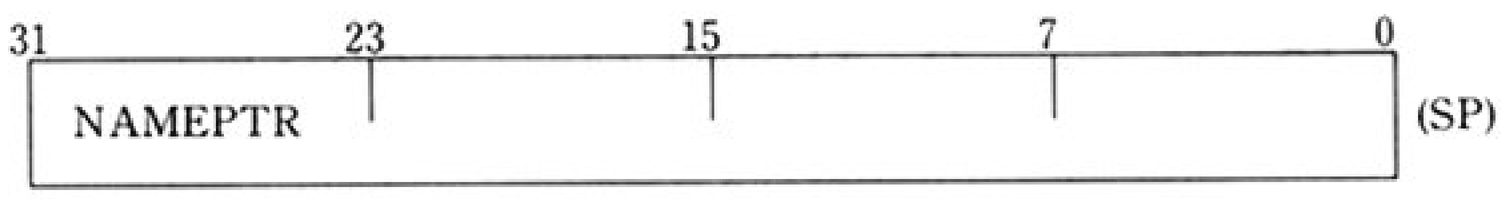
● chdir (NAMEPTR)　　DOS コール番号：$FF3B

機　　能　カレントディレクトリを変更します。

引　　数　NAMEPTR (r)　address　　文字列（ディレクトリ名）へのポインタ

コール

```
            PEA         NAMEPTR
            DC. W       _CHDIR
            ADDQ. L     #4, SP
              :
NAMEPTR:    DC. B       'A:¥DOC', 0
```



リターン　D0. L

　D0. L ＜ 0……エラーコード

解　　説　カレントディレクトリを、NAMEPTRで指定したディレクトリ名のサブディレクトリに変更します。

NAMEPTRには、実在するディレクトリを指定してください。

リターンのD0. Lレジスタに通常は0が返りますが、特殊デバイスドライバが対象の時は0以外の正の数が返る場合があります。

負の数の場合は、エラーコードを表します。

create

● create (NAMEPTR, ATR)　　DOS コール番号：$FF3C

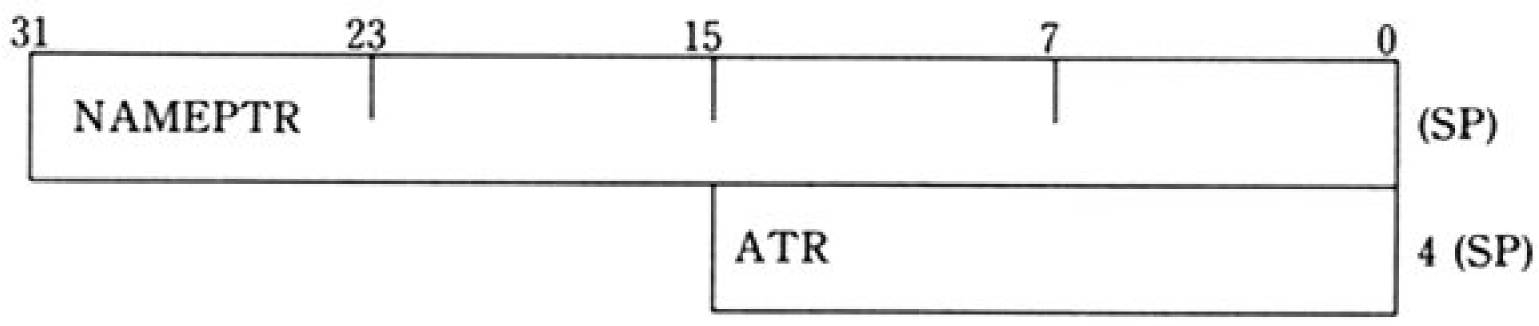
機　　能　ファイルを作成します。

引　　数

NAMEPTR (r)	address	文字列（ファイル名）へのポインタ
ATR(r)	word	ファイル属性

コ ー ル

```
           MOVE.W   ATR, -(SP)
           PEA      NAMEPTR
           DC.W     _CREATE
           ADDQ.L   #6, SP
              :
NAMEPTR:   DC.B     'A:¥DOC¥TEST.DOC', 0
```



リターン　D0. L　　作成したファイルのファイルハンドル

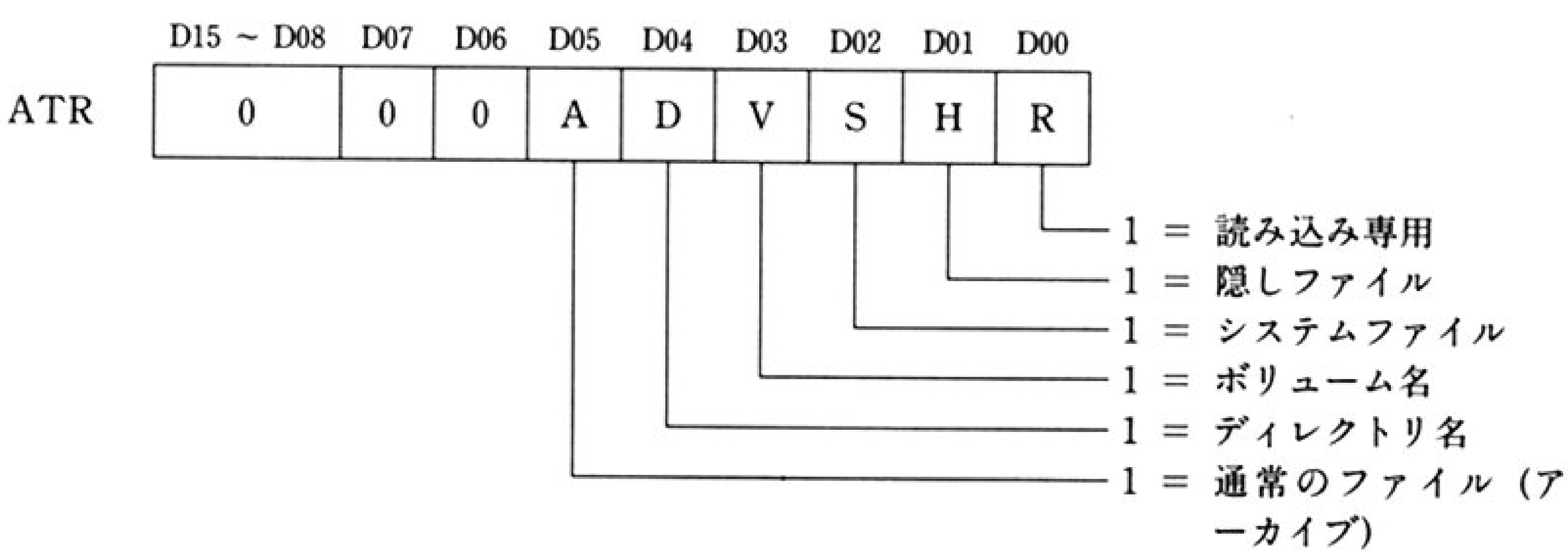
D0. L ＜ 0……エラーコード

解　　説　新しいファイルを作成します。

NAMEPTRで指定したファイル名がすでに存在している場合には、前のファイルの内容は失われます。

ATRには、作成するファイルの属性を指定します。

各ビットは次の通りです。



ファイルが正常に作成できたら、リターン値（D0. L）に作成したファイルのファイルハンドルを返します。

また、負の数ならエラーコードです。

open

● open (NAMEPTR, MODE)　　DOSコール番号：$FF3D

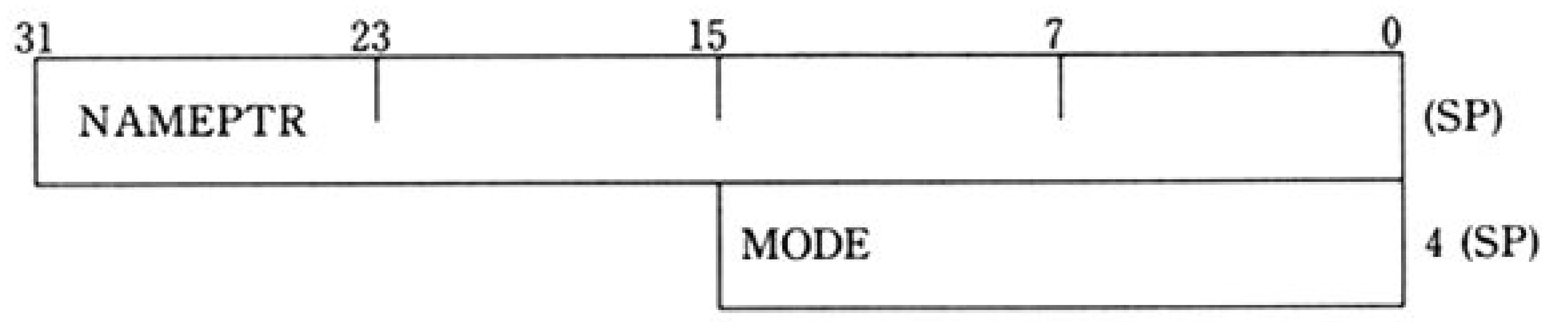
機　能　ファイルをオープンします

引　数

NAMEPTR (r)	address	文字列（ファイル名）へのポインタ
MODE (r)	word	アクセスモード

コール

```
          MOVE. W   MODE, -(SP)
          PEA       NAMEPTR
          DC. W     _OPEN
          ADDQ. L   #6, SP
             ⋮
NAMEPTR:  DC. B     'A:¥DOC¥TEST. DOC', 0
```



リターン　D0. L　オープンしたファイルのファイルハンドル

D0. L ＜ 0……エラーコード

解　説　NAMEPTRで指定したファイルをオープンします。

ファイルのアクセス方法は、MODEで指定します。

MODEのフォーマットを以下に示します。

ビット8：辞書アクセスの指定

1…辞書アクセスモード

辞書用の特殊なファイルハンドルを返す

ユーザは使用禁示

0…通常のファイリングモードの指定

ビット6〜4：シェアリングモードの指定

000…互換モード

001…読み込み／書き込みの両方の禁止

010…書き込みの禁止

011…読み込みの禁止

100…許可

ビット1〜0：読み書きの指定

00…読み込みモードでオープン

01…書き込みモードでオープン

10…読み込み／書き込みのモードでオープン

ビット6～4で指定するシェアリングモードとは、複数のプロセスによるファイルの共有を規制するものです。

たとえば、シェアリングモードに001（読み込み／書き込みの両方の禁止）を指定してオープンされたファイルに対しては、そのファイルをオープンしたプロセス以外から、読み込み／書き込みの両方ができなくなります。

複数のプロセスが並列に実行されている場合、1つのファイルに対して複数のプロセスが同時に読み込み／書き込みを行う可能性があります。

このため、適切なシェアリングモードを指定して、ファイルアクセスの排他処理を行う必要があります。

ファイルが正常にオープンされると、リターン値（D0. L）としてファイルハンドルを返します。

また、負の数ならエラーコードです。

close

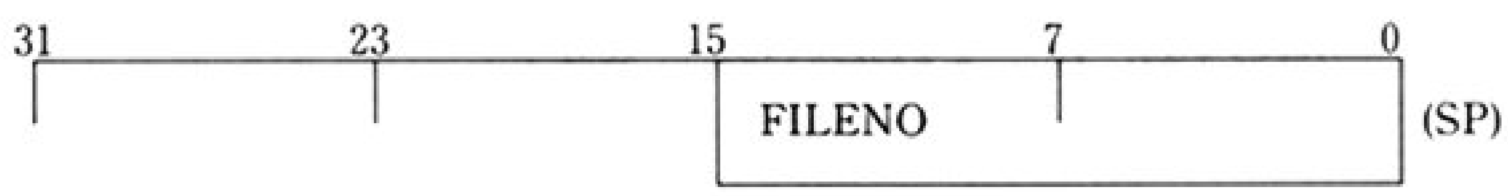
● close (FILENO)　　DOS コール番号：$FF3E

機　　能　ファイルをクローズします。

引　　数　FILENO (r)　word　ファイルハンドル

コ　ー　ル

```
MOVE. W   FILENO, -(SP)
DC. W     _CLOSE
ADDQ. L   #2, SP
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　　説　FILENOで指定したファイルハンドルをクローズします。

read

● read (FILENO, DATAPTR, SIZE)　　　DOS コール番号：$FF3F

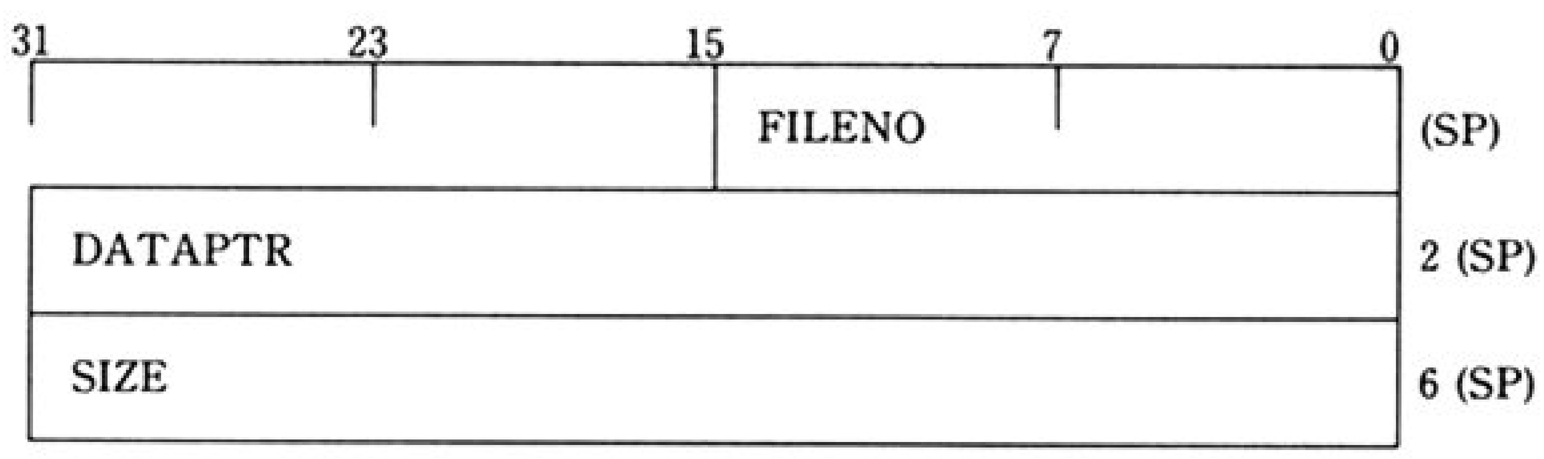
機　能　ファイルから読み込みます。

引　数

FILENO (r)	word	ファイルハンドル
DATAPTR (w)	address	読み込むバッファのポインタ
SIZE (r)	long	読み込むデータの長さ

コール

```
            MOVE. L   SIZE, -(SP)
            PEA       DATAPTR
            MOVE. W   FILENO, -(SP)
            DC. W     _READ
            LEA       10 (SP), SP
              :
DATAPTR:    DS. B     ???        *???は読み込むデータの長さ
```



リターン　D0. L　　実際に読み込んだバイト数

D0. L ＜ 0……エラーコード

解　説　FILENOで指定したファイルハンドルから、SIZEで指定する長さのデータをDATAPTRで指定するバッファに読み込みます。

読み込み操作は、指定したファイルと結合したファイルポインタの現在位置から始めます。

読み込み操作の後には、ポインタはまだ読んでいない次の文字を指します。

スタックに10バイト使用するため、ここではADDQ. Lでスタックを戻せません。

LEA命令で戻してください。

リターンのD0. Lには、実際に読み込んだバイト数を返します。

負の数ならばエラーコードです。

実際に読み込んだバイト数が指定したSIZEより小さい場合は、ファイルを終わりまで読み込んだ場合です。

write

● write (FILENO, DATAPTR, SIZE)　　　DOS コール番号：$FF40

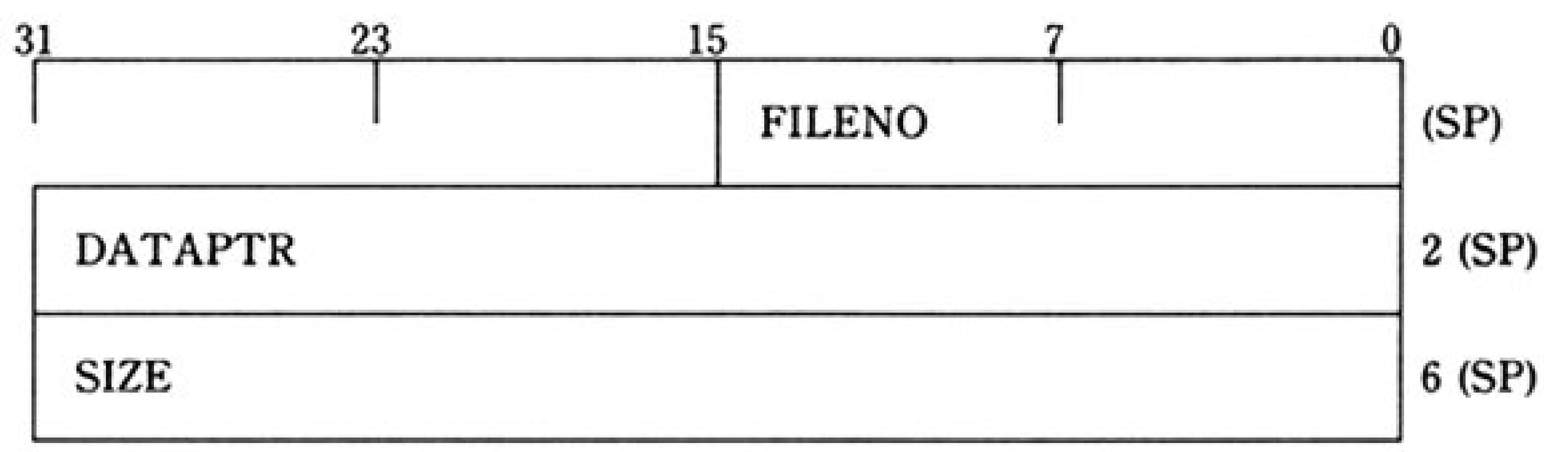
機　　能　ファイルへ書き込みます。

引　　数

FILENO (r)	word	ファイルハンドル
DATAPTR (r)	address	書き込むデータのポインタ
SIZE (r)	long	書き込むデータの長さ

コール

```
          MOVE. L   SIZE, -(SP)
          PEA       DATAPTR
          MOVE. W   FILENO, -(SP)
          DC. W     _WRITE
          LEA       10 (SP), SP
             ⋮
DATAPTR:  DC. B     DATA, DATA, DATA, ……
```



リターン　D0. L　　実際に読み込んだバイト数

D0. L ＜ 0……エラーコード

解　　説　SIZEで指定した長さのデータを、DATAPTRで指定したバッファからFILENOで指定したファイルハンドルに書き込みます。

書き込み処理は、指定したファイルのファイルポインタの現在位置から始めます。

書き込んだ後には、ファイルポインタは、実際に書き込んだバイト数だけ進みます。

スタックに10バイト使用するため、ADDQ. Lではスタックを戻せません。

LEA 命令で戻してください。

リターンのD0. Lには、実際に書き込んだバイト数を返します。

負の数ならばエラーコードです。

実際に書き込んだバイト数が指定したSIZEより小さい場合、ディスクの空きが足りなくなった場合です。

delete

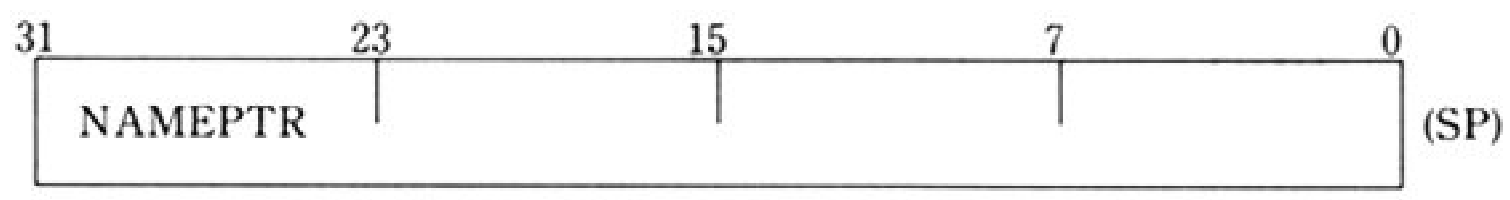
● delete (NAMEPTR)　　DOS コール番号：$FF41

機　能　ファイルを削除します。

引　数　NAMEPTR (r)　address　　文字列（ファイル名）のポインタ

コール

```
            PEA       NAMEPTR
            DC. W     _DELETE
            ADDQ. L   #4, SP
              ⋮
NAMEPTR:    DC. B     'A:¥TEST. DOC', 0
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　説　NAMEPTR で指定したファイルを削除します。

ワイルドカードの指定や、ディレクトリ名の指定はできません。

seek

● seek (FILENO, OFFSET, MODE)　　　DOSコール番号：$FF42

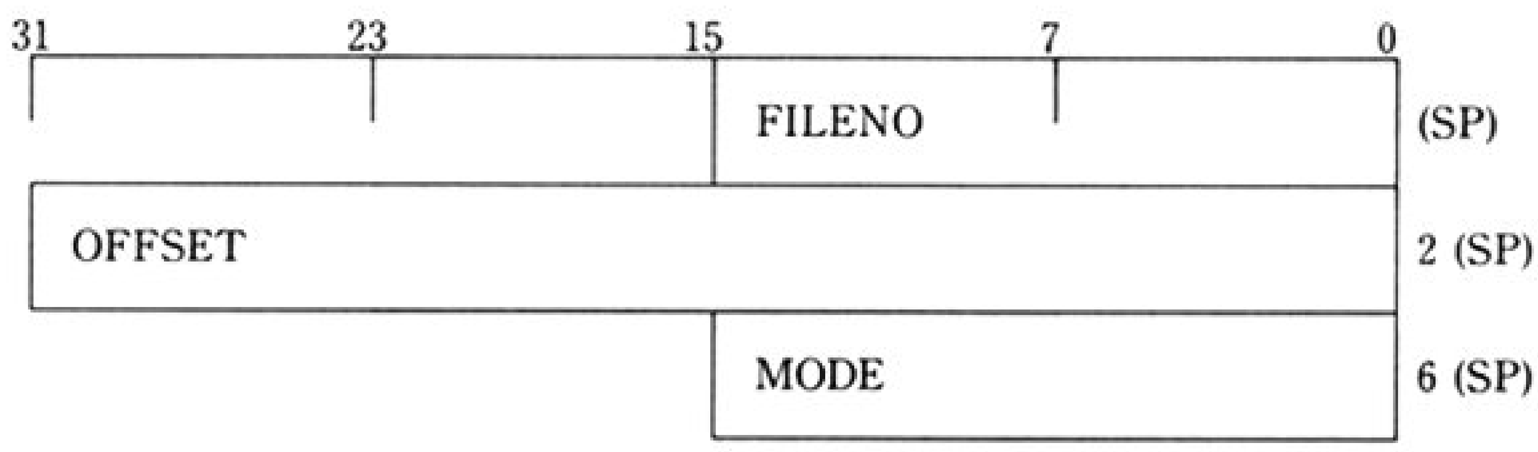
機　　能　ファイルポインタを移動します。

引　　数

FILENO (r)	word	ファイルハンドル
OFFSET (r)	long	オフセット
MODE (r)	word	モード

コール

```
MOVE. W   MODE, -(SP)
MOVE. L   OFFSET, -(SP)
MOVE. W   FILENO, -(SP)
DC. W     _SEEK
ADDQ. L   #8, SP
```



リターン　D0. L　　　現在の先頭からのオフセット

D0. L ＜ 0……エラーコード

解　　説　FILENOで指定したファイルハンドルのランダムファイルと、結合したファイルポインタを、各MODEで指定する位置から、OFFSETで指定したバイト数だけ移動します。

MODE ＝ 0……先頭からのオフセット

MODE ＝ 1……現在位置からのオフセット

MODE ＝ 2……ファイルエンドからのオフセット

次のファイルは操作は、移動した新しい位置から始められます。

seekはファイルのどこにでもポインタを移動することができます。

ポインタをエンドオブファイルを超えて移動するかまたは、ファイルの先頭よりも前へ移動するとエラーとなります。

リターンのD0. Lには、現在の先頭からのオフセットを返します。

負数ならエラーコードです。

chmod

● chmod (NAMEPTR, ATR)　　DOS コール番号：$FF43

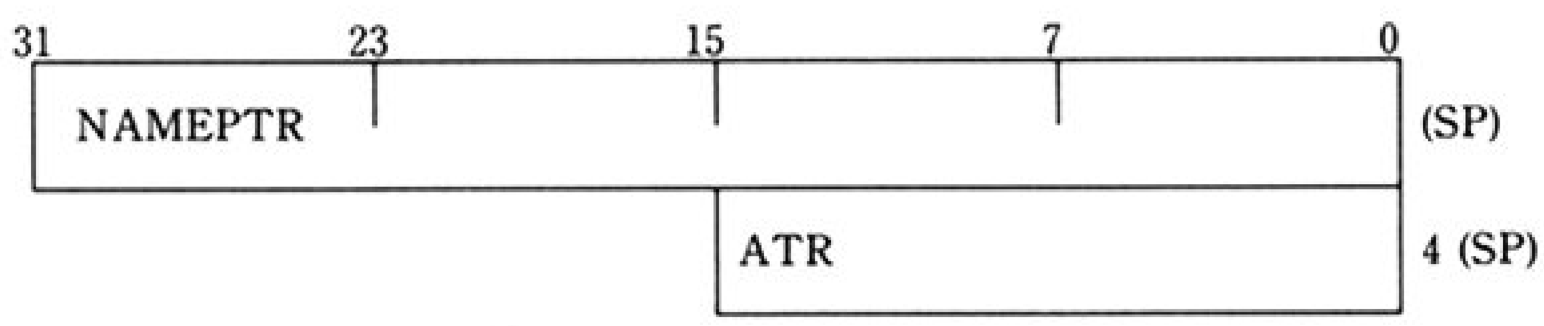
機　能　ファイルの属性を変更します。

引　数

NAMEPTR (r)	address	文字列（ファイル名）のポインタ
ATR (r)	word	ファイル属性

コール

```
          MOVE. W  ATR, -(SP)
          PEA      NAMEPTR
          DC. W    _CHMOD
          ADDQ. L  #6, SP
             :
NAMEPTR:  DC. B    'A : ¥DOC¥TEST. DOC', 0
```

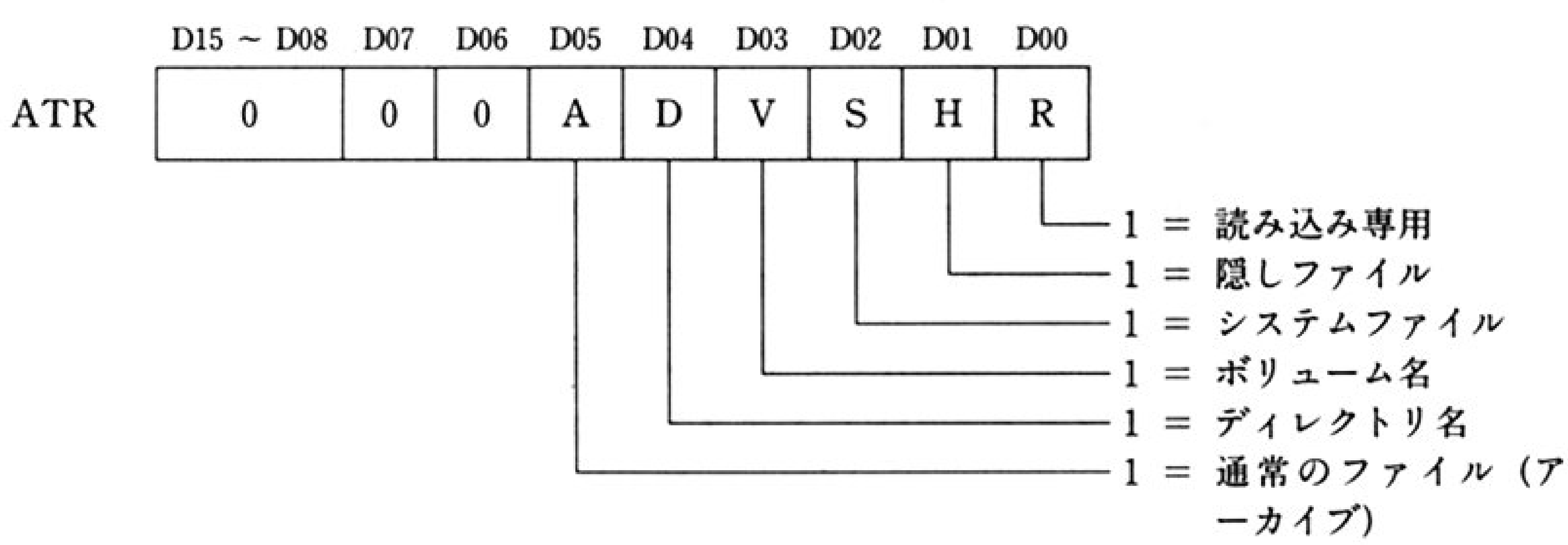


リターン　D0. L　　指定したファイルの属性

D0. L ＜ 0……エラーコード

解　説　NAMEPTR で指定したファイルの属性を、ATR にしたがって変更します。

ATR の内容は次の通りです。



ATR に−1 を指定すると、指定したファイルの属性を D0. L に返します。

負の数ならばエラーコードです。

ioctrl

●ioctrl (MD, [FILENO] [DRIVE] [COUNT, TIME] [, DT] [, PTR, LEN] [, F_CODE, PTR])　　DOS コール番号：$FF44

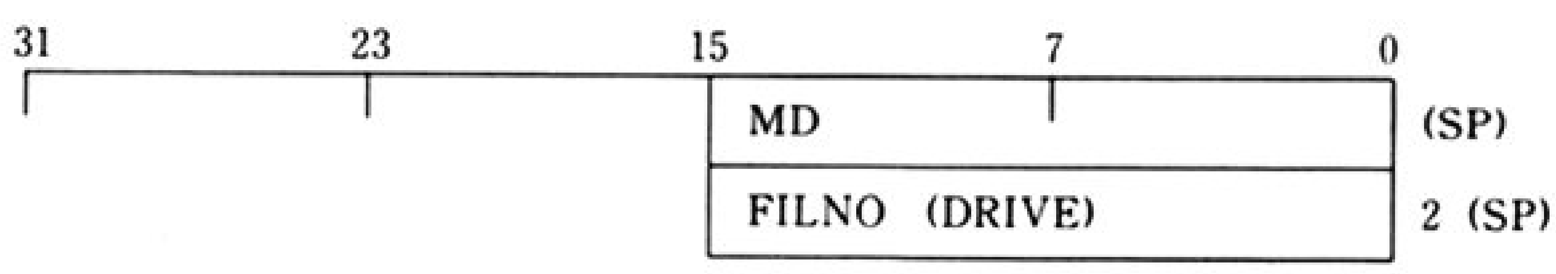
機　能	デバイスドライバの IOCTRL によりダイレクトに入出力します。

引　数

MD (r)	word	コントロールモード（この値によって引数が異なります）
FILENO (r)	word	ファイルハンドル
DT (r)	word	装置情報
DRIVE (r)	word	装置番号
PTR (r)	address	ポインタ
LEN (r)	long	読み書きバイト数
COUNT (r)	word	リトライ回数
TIME (r)	word	時間
F_CODE (r)	word	ファンクション番号

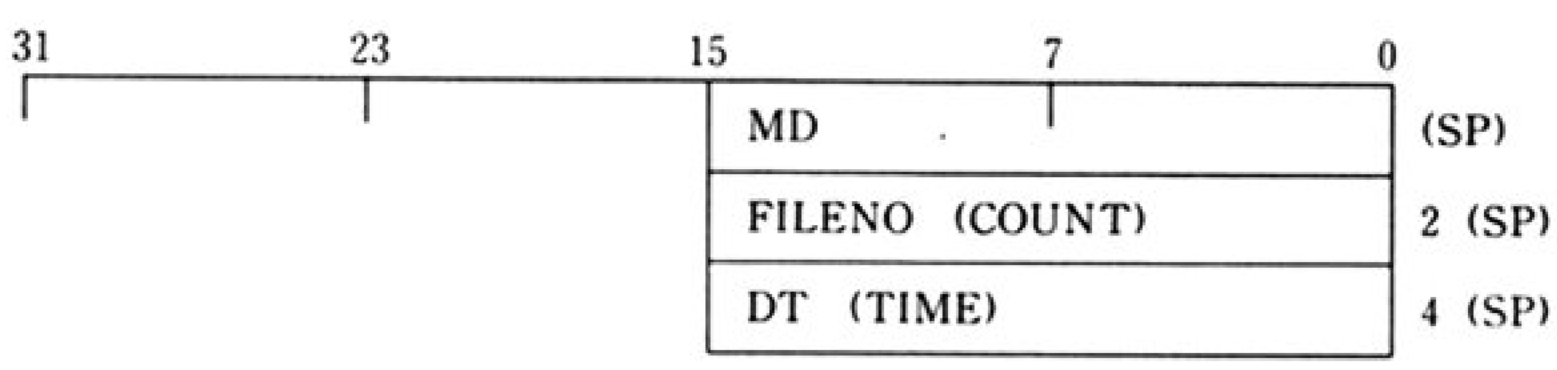
コール　〈MD = 0, 6, 7, 9, 10〉

```
MOVE.W   FILENO, -(SP)   *または MOVE.W  DRIVE, -(SP)
MOVE.W   MD, -(SP)
DC.W     _IOCTRL
ADDQ.L   #4, -(SP)
```

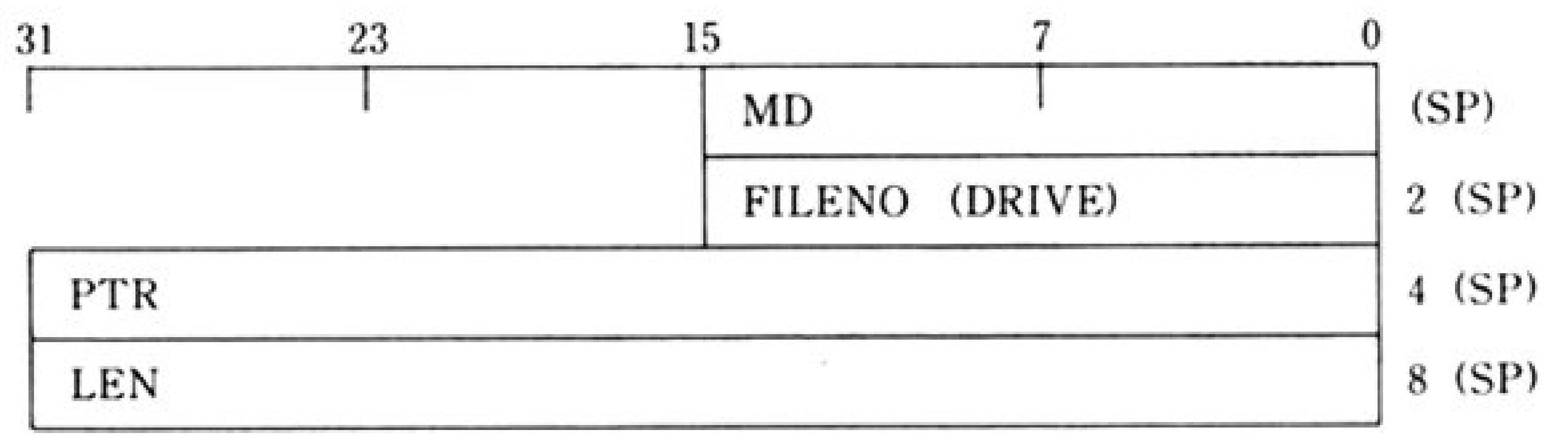


〈MD = 1, 11〉

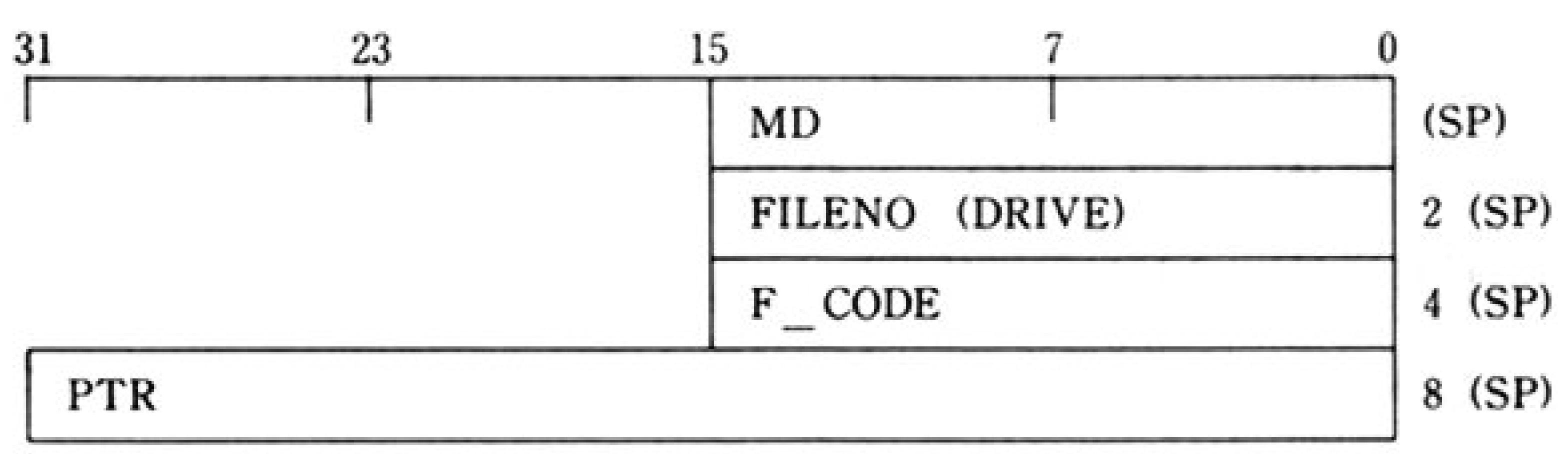
```
MOVE.W   DT, -(SP)       *または MOVE.W  TIME, -(SP)
MOVE.W   FILENO, -(SP)   *       MOVE.W  COUNT, -(SP)
MOVE.W   MD, -(SP)
DC.W     _IOCTRL
ADDQ.L   #6, -(SP)
```



```
〈MD = 2, 3, 4, 5〉
        MOVE.L   LEN, -(SP)
        PEA      PTR
        MOVE.W   FILENO, -(SP)    *または MOVE.W  DRIVE, -(SP)
        MOVE.W   MD, - (SP)
        DC.W     _IOCTRL
        LEA      12(SP), SP
          :
PTR:    DS.B     ???              *???は読み書きバイト数
```




```
〈MD = 12, 13〉
        PEA      PTR
        MOVE.W   F_CODE, -(SP)
        MOVE.W   FILENO, -(SP)    *または MOVE.W  DRIVE, -(SP)
        MOVE.W   MD, -(SP)
        DC.W     _IOCTRL
        ADDQ.L   10(SP), -(SP)
          :
PTR:    DS.B     ???              *???はデバイスドライバによる
```



リターン

〈MD = 0, 1〉
D0.L　装置情報

〈MD = 2〜5〉
D0.L　実際に読み書きしたバイト数

〈MD = 6, 7, 9, 10〉
D0.L　ステータス

〈MD = 11, 12, 13〉
デバイスドライバによります。

解　説

デバイスドライバのダイレクト入出力をします。
MDにより、それぞれ次の意味を持ちます。

MD =0　　ioctrl (0, FILENO)
FILENOで指定したハンドルの装置情報を、D0.Lに返します。

MD = 1　　ioctrl (1, FILENO, DT)
FILENOで指定したハンドルに、装置情報を設定します。
D0.Lには、前の装置情報を返します。

MD = 2　　ioctrl (2, FILENO, PTR, LEN)
FILENOで指定したハンドルから、PTRにLENバイト読み込みます。
D0.Lには、読み込みバイト数を返します。

MD = 3　　ioctrl (3, FILENO, PTR, LEN)
FILENOで指定したハンドルに、PTRからLENバイト書き込みます。
D0.Lには、書き込みバイト数を返します。

MD = 4　　ioctrl (4, DRIVE, PTR, LEN)
DRIVEで指定した装置番号 (0 =カレント、1 = A:) から、PTRにLENバイト読み込みます。
D0.Lには、読み込みバイト数を返します。

MD = 5　　ioctrl (5, DRIVE, PTR, LEN)
DRIVEで指定した装置番号 (0 =カレント、1 = A:) に、PTRからLENバイト書き込みます。
D0.Lには、書き込みバイト数を返します。

MD = 6　　ioctrl (6, FILENO)
FILENOで指定するハンドルの入力ステータスをD0.Lに返します。
内容は次の通りです。

D0.L = $FF……可
D0.L = $00……不可

MD = 7　　ioctrl (7, FILENO)

FILENO で指定するハンドルの出力ステータスを、D0.L に返します。

内容は次の通りです。

D0.L = $FF……可

D0.L = $00……不可

MD = 9　　ioctrl (9, DRIVE)

DRIVE で指定するドライブが、ローカルかリモートかを、D0.L に返します。

内容は次の通りです。

D0.L……ローカルならドライブ情報

D0.L……リモートならビット 12 が 1

D0.L < 0……エラー

MD = 10　　ioctrl (10, FILENO)

FILENO で指定するドライブが、ローカルかリモートかを、D0.L に返します。

内容は次の通りです。

D0.L……ローカルならドライブ番号

D0.L……リモートならビット 12 が 1

D0.L < 0……エラー

MD = 11　　ioctrl (11, COUNT, TIME)

リトライ数を設定します。デフォルトの値は 3 回で、時間は 100 秒です。

この関数はデバイスドライバでサポートされます。

MD = 12　　ioctrl (12, FILENO, F_CODE, PTR)

ハンドルによる特殊コントロールを行います。

MD = 13　　ioctrl (13, DRIVE, F_CODE, PTR)

ドライブによる特殊コントロールを行います。

※デバイスドライバのデバイスヘッダの IOCTRL ビットが 0 の場合は、入出力できません。

装置の情報の値は、次のような意味を持っています。

ビット	意味	
0	標準入力	
1	標準出力	
2	NULL	
3	CLOCK	
4	(予備)	
5	0……COOKED	1……RAW
6	(予備)	
7	0……ドライブ	1……デバイス
8〜11	(予備)	
12	0……ローカル	1……リモート
13	0……通常のブロックデバイス	1……特殊ブロックデバイス
14	0……IOCTRL 不可	1……IOCTRL 可

dup

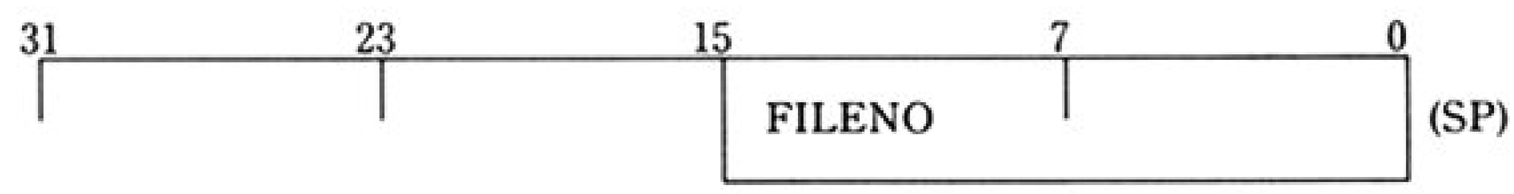
● dup (FILENO)　　DOSコール番号：$FF45

機　能	ファイルハンドルを複写します。

引　数　FILENO (r)　word　ファイルハンドル

コール

```
MOVE. W    FILENO, -(SP)
DC. W      _DUP
ADDQ. L    #2, SP
```



リターン　D0. L　新しいファイルハンドル

D0. L ＜ 0……エラーコード

解　説　現在のオープンされているファイルハンドル（FILENO）を複写して、2つ目のファイルハンドルを作ります。

同一ファイルのファイルハンドルは、同じファイルポインタを使用するので、どちらのファイルハンドルを使ってもファイルの操作ができるようになります。

リターンのD0. Lには、複写された新しいファイルハンドルを返します。

負の数ならばエラーコードです。

dup2

● dup2 (FILENO, NEWNO)　　DOS コール番号：$FF46

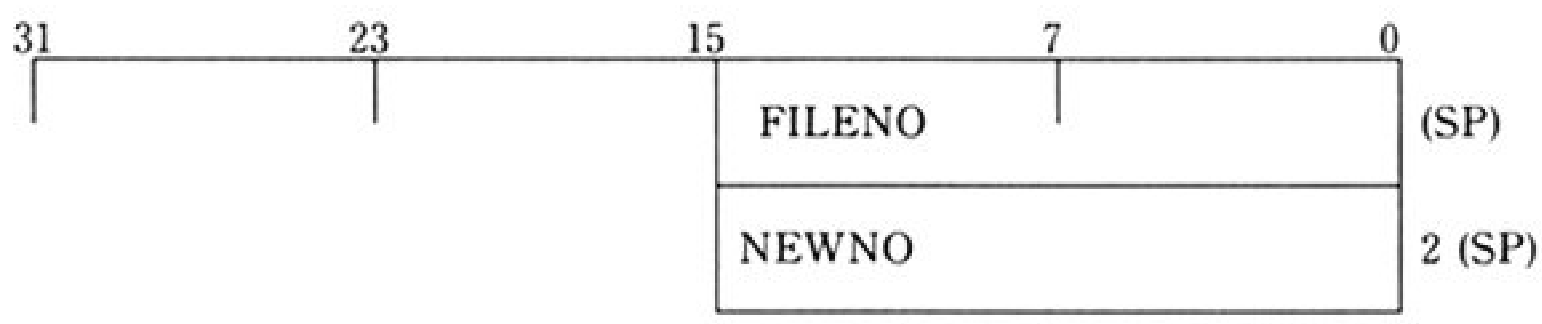
機　能　ファイルハンドルを強制複写します

引　数

FILENO (r)	word	ファイルハンドル
NEWNO (r)	word	ファイルハンドル

コール

```
MOVE. W   NEWNO, −(SP)
MOVE. W   FILENO, −(SP)
DC. W     _DUP2
ADDQ. L   # 4, SP
```



リターン　D0. L

　D0. L ＜ 0……エラーコード

解　説　現在オープンされているファイルハンドル (FILENO)を強制的に複写して、2つ目のファイルハンドル (NEWNO)を作ります。

NEWNOで指定したファイルハンドルが、すでにオープンされているときには、自動的にそれをクローズしてから複写します。

同一のファイルのファイルハンドルは、同じファイルポインタを使用するので、どちらのファイルハンドルを使ってもファイルの操作ができるようになります。

リターンのD0. Lが負の数ならばエラーコードを表しています。

curdir

● curdir (DRIVE, PATHBF)　　　DOS コール番号：$FF47

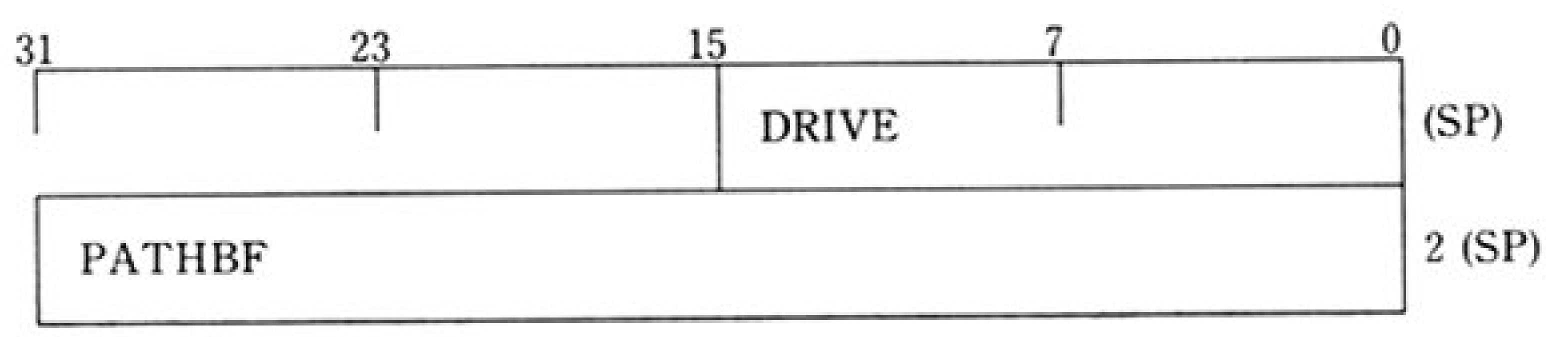
機　能　カレントパスを得ます。

引　数

DRIVE (r)　word　ドライブ番号
PATHBF (w)　address　バッファのポインタ

コール

```
         PEA       PATHBF
         MOVE. W   DRIVE, -(SP)
         DC. W     _CURDIR
         ADDQ. L   #6, SP
              ⋮
PATHBF:  DS. B     65        * 65バイト必要
```



リターン

D0. L
　D0. L ＜ 0……エラーコード

解　説

DRIVEで指定したドライブのカレントのパス名を、PATHBFで指定したバッファに書き込みます。
PATHBFは65バイト必要です。
ドライブ番号は次のように指定します。
　DRIVE ＝ 0 ……カレントドライブ
　DRIVE ＝ 1 …… Aドライブ
　DRIVE ＝ 2 …… Bドライブ
バッファに書き込まれるデータは次のようになります。

```
PATHBF:   DC. B   0           *ルートディレクトリの場合
PATHBF:   DC. B   'DOC', 0    *¥DOCサブディレクトリの場合
```

リターンのD0. Lが負数ならエラーコードを表しています。

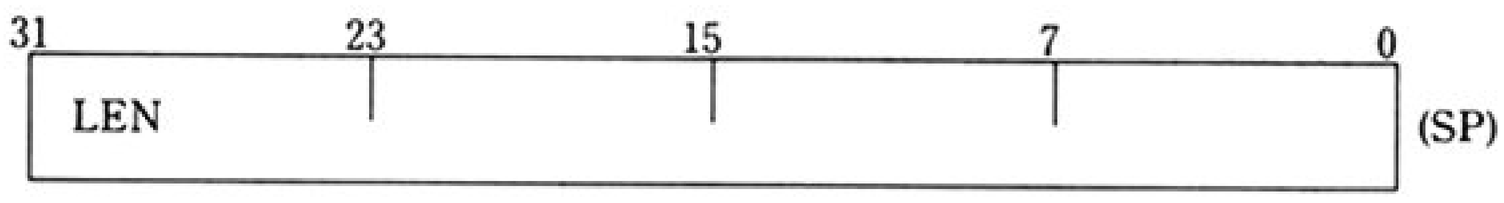
malloc

● malloc (LEN)　　DOS コール番号：$FF48

機　能	メモリを確保します。

引　数	LEN (r)	long	バイト数

コール	MOVE. L	LEN, −(SP)
	DC. W	_MALLOC
	ADDQ. L	#4, SP



リターン	D0. L	確保したメモリブロックのポインタ

D0. L ＝ $81??????……確保できない（$??????は最大バイト数）

D0. L ＝ $8200000?……完全に確保不能（?は不定）

解　説

LEN で指定したバイト数のメモリを確保して、そのポインタを返します。
LEN は、下位 24 ビットのみ有効です。
リターンの D0. L は、確保したメモリブロックのポインタ（メモリ管理ポインタ＋$10）を返します。
確保できない場合は、$81000000＋最大バイト数を返します。
$8200000?を返した場合は、完全に確保不能です（?は不定）。

mfree

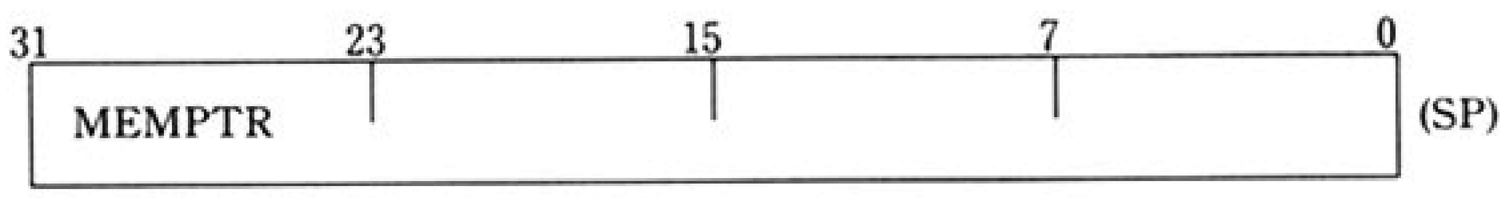
● mfree (MEMPTR)　　DOS コール番号：$FF49

機　能　メモリブロックを開放します。

引　数　MEMPTR (r)　address　メモリブロックのポインタ

コール

```
PEA     MEMPTR
DC. W   _MFREE
ADDQ. L #4, SP
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　説　MEMPTR で指定したメモリブロックを開放します。

MEMPTR は、malloc（$FF48）で返された D0. L の値（確保したメモリブロックのポインタ）です。

不正な MEMPTR を指定するとエラーになります。

MEMPTR ＝ 0 の場合、自分が確保したメモリをすべて開放します。

子プロセスも、子プロセスが確保したメモリもすべて開放します。

リターンの D0. L はエラーコードです。

setblock

● setblock (MEMPTR, NEWLEN)　　　DOS コール番号：$FF4A

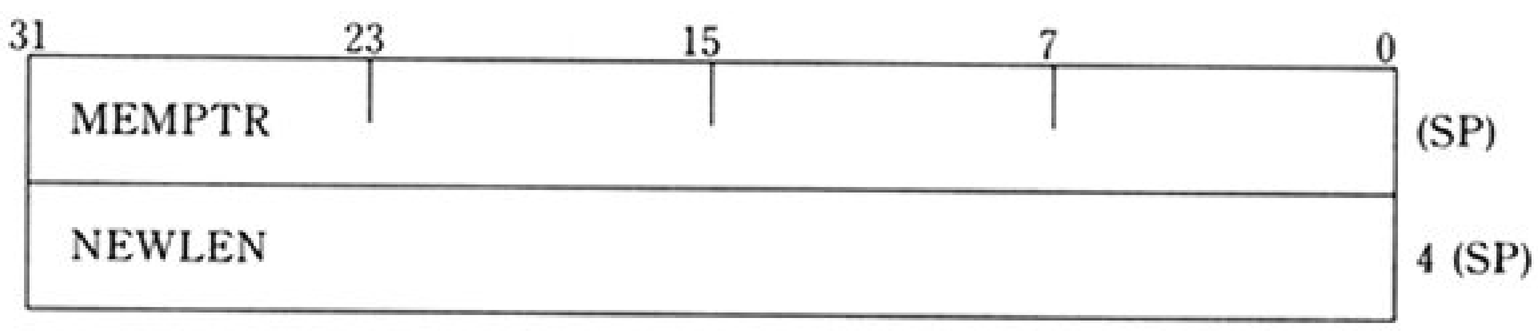
機　　能　メモリブロックを変更します。

引　　数

MEMPTR (r)	address	メモリブロックのポインタ
NEWLEN (r)	long	バイト数

コ ー ル

```
MOVE. L   NEWLEN, -(SP)
PEA       MEMPTR
DC. W     _SETBLOCK
ADDQ. L   #8, SP
```



リターン　D0. L

D0. L ＜ 0………………エラーコード
D0. L ＝ $81?????? ……変更できない（$??????は最大バイト数）
D0. L ＝ $8200000?……完全に変更不可能（?は不定）

解　　説　MEMPTR で指定したメモリブロックを変更します。
メモリ領域の拡大／縮小もできます。
NEWLEN は、下位 24 ビットのみ有効です。
MEMPTR は、malloc（$FF48）で求めた D0 の値（確保したメモリブロックのポインタ）です。
不正な MEMPTR を指定するとエラーになります。
リターンの D0. L はエラーコードです。
変更できない場合は$81000000＋最大バイト数を返します。
$8200000?を返した場合は、完全に変更不能です（?は不定）。

exec

● exec (MD, FIL ［＋TYPE ＊$1000000］ ［P1, P2］) DOSコール番号：$FF4B

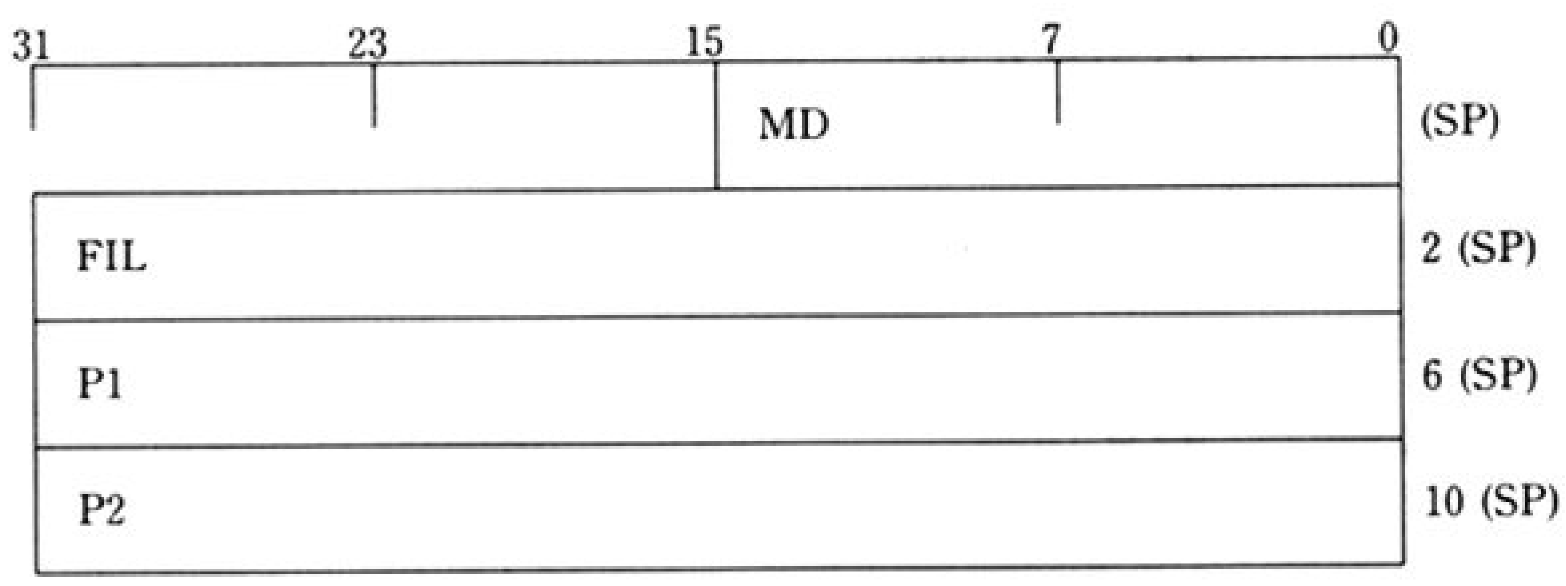
機　　能　プログラムをロード／実行します。

引　　数

MD (r)	word	コントロールモード
FIL (w)	address	文字列（ファイル名）のポインタ
P1 (w)	address	コマンドラインのポインタ
P2 (r)	address	環境ポインタ
TYPE (r)	byte	ファイルタイプ

コ ー ル　<MD = 0, 1>

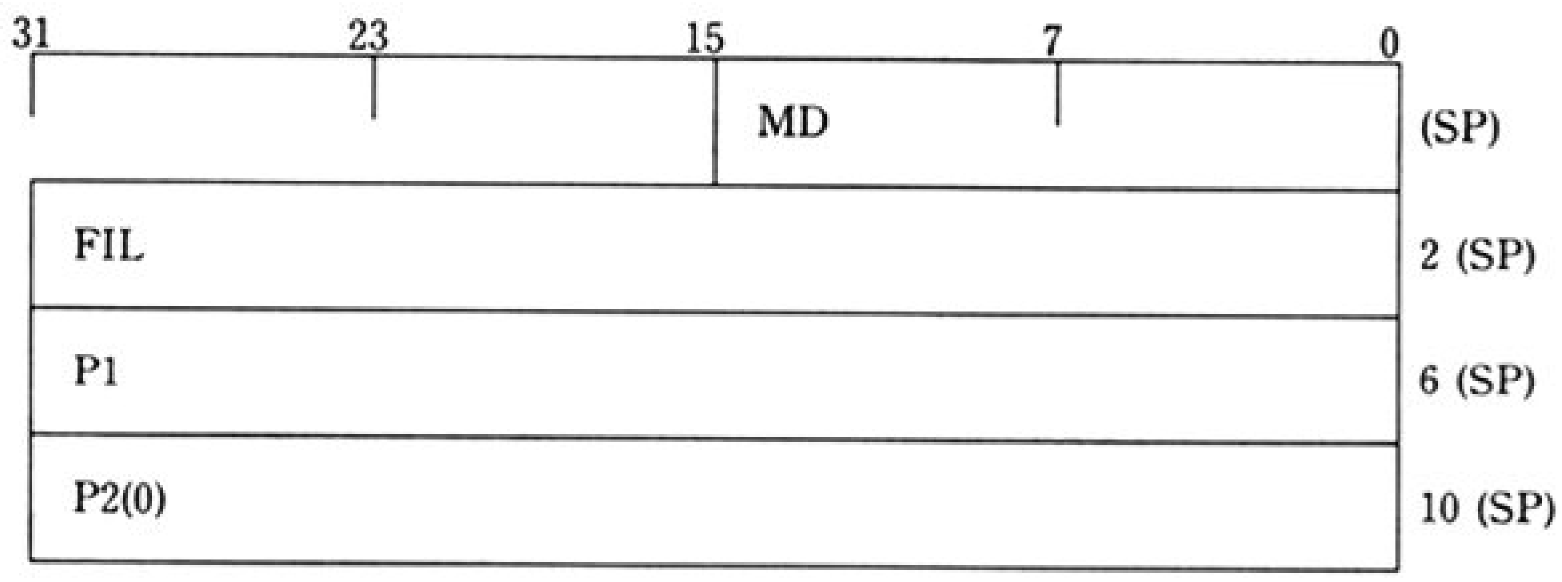
```
        CLR. L      -(SP)              *自分の環境ポインタを指定
        PEA         P1                 *コマンドラインのポインタ
        PEA         FIL                *ファイル名のポインタ
        MOVE. W     MD, -(SP)
        DC. W       _EXEC
        LEA         14(SP), SP
           :
FIL:    DC. B       A:¥BIN¥ED. X       *ファイル名
P1:     DC. B       11                 *コマンドラインの文字数
        DC. B       'DOSCALL. DOC', 0  *コマンドライン
```



<MD = 2>

```
        CLR. L      -(SP)              *自分の環境ポインタを指定
        PEA         P1                 *コマンドライン格納領域
        PEA         FIL                *コマンド行のポインタ
        MOVE. W     MD, -(SP)
        DC. W       _EXEC
```

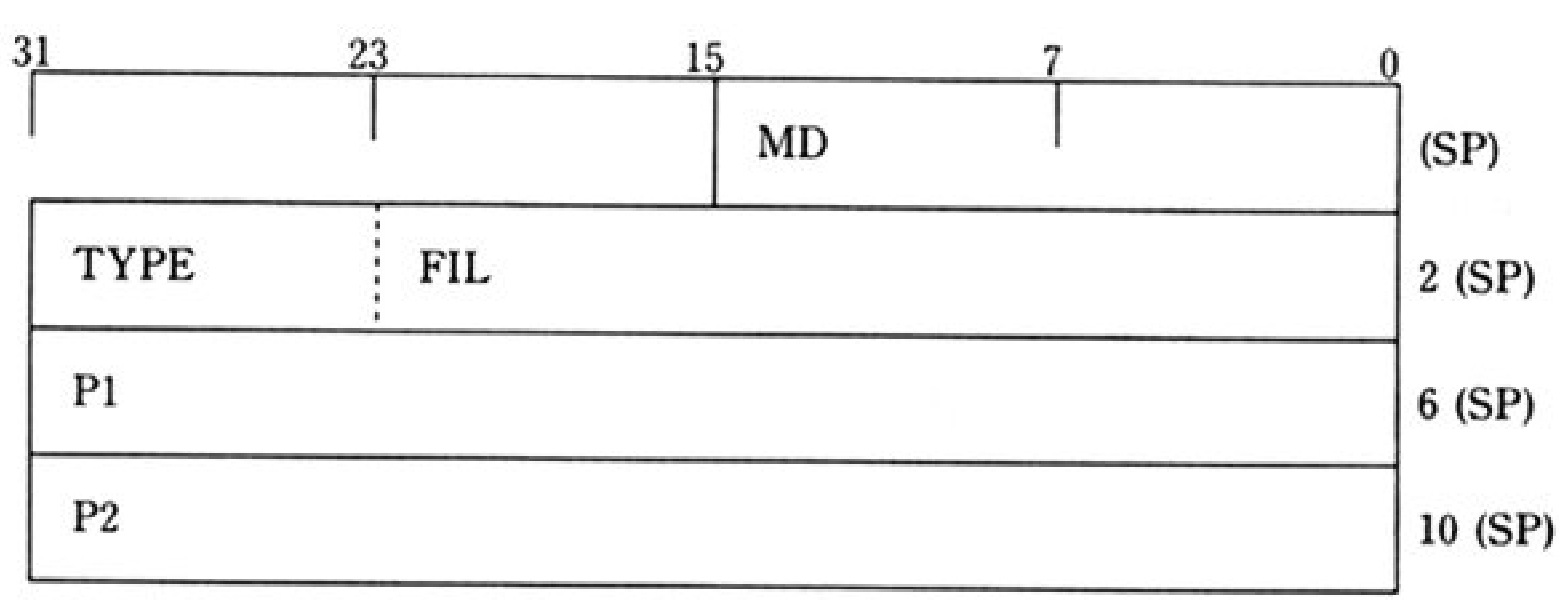
```
        LEA         14 (SP), SP
          ⋮
FIL:    DC. B       'AS  ／t  D:¥TMP  sample', 0
        DS. B       90
P1:     DS. B       256
```



〈MD ＝ 3〉

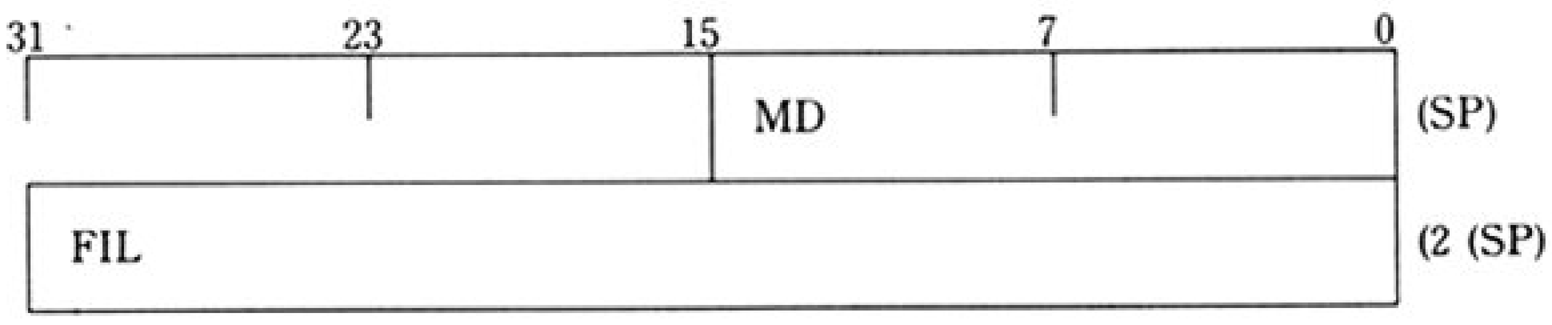
```
TYPEEQU         3                   *.R : 1, .Z : 2, .X : 3

        PEA         P2              *リミットアドレス
        PEA         P1              *ロードアドレス
        PEA         FIL             *ファイル名のポインタ
        MOVE. B     # TYPE, (SP)    * Xタイプのファイルとしてロード
        MOVE. W     MD, −(SP)
        DC. W       _EXEC
        LEA         14 (SP), SP
          ⋮
FIL:    DC. B       'A : ¥BIN¥LK. X', 0
        .EVEN
P1:     DS. B       32*1024
P2:
```



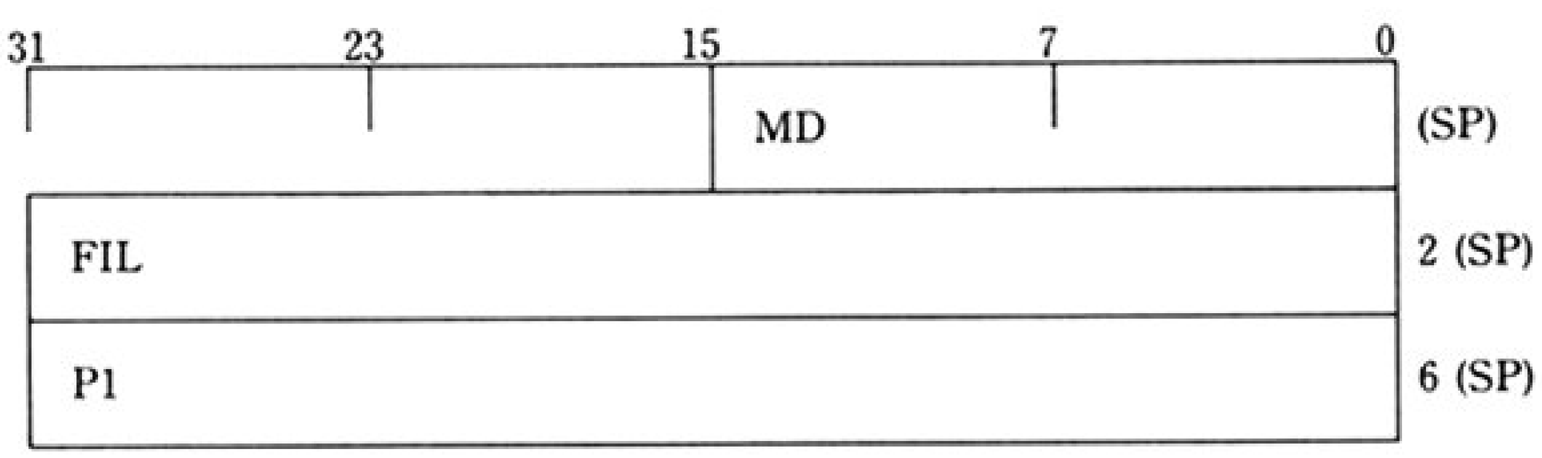
〈MD ＝ 4〉

```
        PEA         FIL                     *実行アドレス
        MOVE.W      MD,-(SP)
        DC.W        _EXEC
        LEA         6(SP),SP
```



〈MD ＝ 5〉

```
        PEA         P1
        PEA         FIL
        MOVE.W      MD,-(SP)
        DC.W        _EXEC
        LEA         10(SP),SP
            :
FIL:    DC.B        'B=¥sample.x',0
P1:     DC.B        'sample1.x',0
```



リターン

〈MD ＝ 0, 4〉

D0.L　プロセス終了コード

D1.L ＜ 0……エラーコード

D1.L～D7.L　プロセス終了なら不定

A1.L～A6.L　プロセス終了なら不定

〈MD ＝ 1〉

D0.L　プログラムの実行アドレス

D0.L ＜ 0……エラーコード

A0.L　メモリ管理ポインタ

A1.L　プログラムの終わり

A2. L　　コマンドライン

A3. L　　環境ポインタ

A4. L　　実行アドレス

<MD ＝ 2>

D0. L

D0. L ＜ 0……エラーコード

<MD ＝ 3>

D0. L　　プログラムの長さ

D0. L ＜ 0……エラーコード

<MD ＝ 5>

D0. L　　モジュール番号

D0. L ＝ $??00……$??はモジュール番号（0～255）

D0. L ＜ 0…………エラーコード

解　　説　FIL で指定したプログラムをロード／実行します。

MD により意味が異なります。

・MD ＝ 0

P1 のコマンドラインと P2 の環境ポインタを指定して、FIL で指定したプログラムをロード／実行します。

P2 の環境ポインタで 0 を指定すると、FIL の環境が自分の環境と同じになります。

リターンの D0. L が正の数の場合はプロセス終了コード、負の数の場合はエラーコードです。

実行されるプログラムから見ると、PSP の各パラメータのほかに、次に示す値がわたされてきます。

PSP は内部的にセットされます。

A0 ＝ PSP－$10（メモリ管理ポインタ）

A1 ＝ DATAEND（プログラムに割りあてられているアドレスの終わり）

A2 ＝ コマンドライン

A3 ＝ 環境ポインタ

A4 ＝ 実行アドレス

・MD ＝ 1

P1 のコマンドラインと P2 の環境ポインタを指定して、FIL で指定するプログラムをロードします。

P2の環境ポインタで0を指定すると、FILの環境が自分の環境と同じになります。

リターンのD0. Lが正の数の場合はロードしたプログラムの実行アドレス、負の数の場合はエラーコードです。

ロードされたあとは、PSPの各パラメータのほかに、次に示す値がリターンします。

PSPは内部的にセットされます。

A0 ＝ PSP－$10（メモリ管理ポインタ）

A1 ＝ DATAEND（プログラムに割りあてられているアドレスの終わり）

A2 ＝ コマンドライン

A3 ＝ 環境ポインタ

A4 ＝ 実行アドレス

・MD ＝ 2

P2の環境からPATHを検索して、FILのコマンド行をプログラム名（ドライブ名・パス名つき）とコマンドラインに分け、FILとP1の各ポインタに設定します。

FILは90バイト以上、P1は256バイト以上必要です。

P2の環境ポインタで0を指定すると、自分の環境を検索します。

MD ＝ 2でexecをコールした後、P1のコマンドラインとFILのファイル名を指定し、MD ＝ 0または1でexecをコールすると、PATHが設定してあるディレクトリ上のプログラムも簡単に実行できます。

・MD ＝ 3

P1のロードアドレスとP2のリミットアドレスを指定して、FILで指定したプログラムをロードします。

リターン値（D0. L）が正の数の場合はプログラムの長さで、負の数の場合はエラーコードです。

・MD ＝ 4

FILは実行アドレスを示します。

MD ＝ 1でロードした後プログラムを実行したいときに使います。

リターンD0. Lが正の数の場合はプロセス終了コード、負の数の場合はエラーコードです。

・MD ＝ 5

FILで指定したオーバーレイファイルから、P1で指定したファイルのモジュール番号を得ます。

リターン値（D0. L）は、ビット15～8にモジュール番号（0～255）を返します。

負の数の場合はエラーコードです。

リターンのD0. Lが負の数ならエラーコード（ロード／実行していない）です。
正の数の場合は、MDによりそれぞれの値が返されます。
実行してきた場合は、D1～D7／A0～A6の内容は保証されません。
MD ＝ 1の場合は、A0～A4にも値が返ります。
・MD ＝ 0、1、3のとき
　TYPE（FILの最上位8ビット）は通常0ですが、次の値で意味を持ちます。
　0……拡張子（. R、. Z、. X）に従い、ロードします。
　1…… . Rタイプのファイルとしてロードします。
　2…… . Zタイプのファイルとしてロードします。
　3…… . Xタイプのファイルとしてロードします。

MDの上位8ビットは通常0ですが、BINDコマンドで作成したオーバーファイル中の各ファイルのロードが実行のため、次の意味を持ちます。
ビット15～8：モジュール番号（0～255）
ビット 7～8：コントロールモード
BINDコマンドの／Lスイッチで次のように出力されたとして、

B>BIND ／L sample. x

＝No＝ ＝NAME＝
　0 sample1. x
　1 sample2. x
　2 sample3. x

それぞれをロードおよび実行させるときは、

sample1. xの場合は、MD ＝ $0000
sample2. xの場合は、MD ＝ $0100
sample3. xの場合は、MI ＝ $0200

とします。

exit2

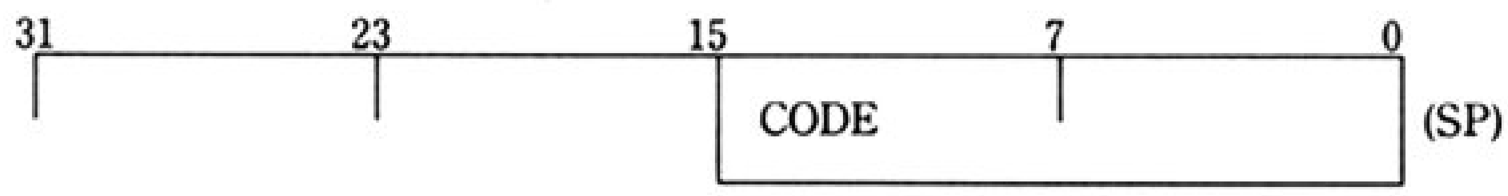
● exit2 (CODE)　　DOS コール番号：$FF4C

機　　能　終了コードを指定して終了します。

引　　数　CODE (r)　　word　　終了コード

コ ー ル

```
MOVE. W   CODE, -(SP)
DC. W     _EXIT2
```



リターン　なし

解　　説　CODE で指定した終了コードをもって、プログラムを終了します。
現在オープンしているファイルハンドルをすべてクローズします。
子プロセスがオープンしたファイルハンドルもクローズします。

wait

● wait ( )　　DOS コール番号：$FF4D

機　能	プロセスの終了コードを得ます。
引　数	なし
コール	DC. W　_WAIT
リターン	D0. L　終了コード
解　説	自分が実行した子プロセスの終了コードを D0. L に返します。

files

● files (FILBUF, NAMEPTR, ATR)　　　DOSコール番号：$FF4E

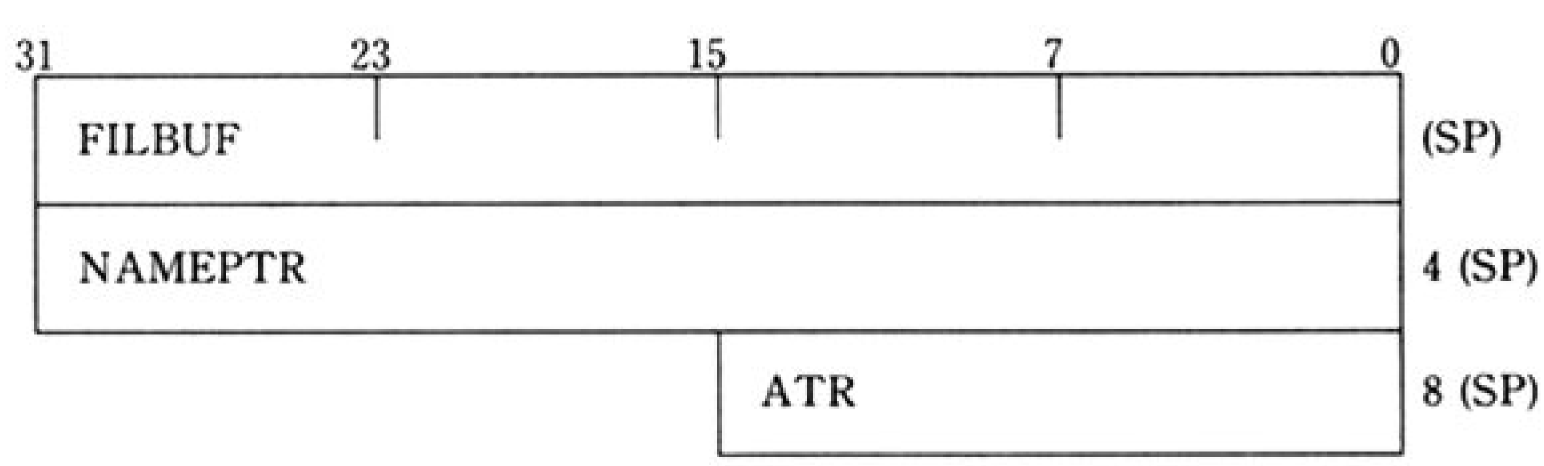
機　能　ファイルを検索し、ファイル情報を得ます（最初のファイル）。

引　数

FILBUF (w)	address	バッファ（ファイル情報）へのポインタ
NAMEPTR (r)	address	文字列（ファイル名）へのポインタ
ATR (r)	word	属性モード

コール

```
            MOVE. W  ATR, -(SP)
            PEA      NAMEPTR
            PEA      FILBUF
            DC. W    _FILES
            LEA      10(SP), SP
               ⋮
NAMEPTR:    DC. B    '*. DOC', 0
FILBUF:     DS. B    1           * ATR          ┐
            DS. B    1           * DRIVENO      │ Human68k
            DS. W    1           * DIRSLS       │ 内部で使用
            DS. W    1           *DIRFAT        ├ 壊すと
            DS. W    1           * DIRCES       │ nfiles($FF4F)
            DS. W    1           * DIRPOS       │ できなくなる
            DS. B    8           *'FILENAME'    ┘
            DS. B    3           *'EXT'
            DS. B    1           *アトリビュートの一致
            DS. W    1           *ファイルの変更最終時刻
            DS. W    1           *ファイルの変更最終月日
            DS. L    1           *ファイルサイズ
            DS. B    23          *ファイル名(主ファイル名+'.'
                                   +拡張子+0)
                                 *計53バイト
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　　説　ATRで指定した属性にあてはまるファイル（NAMEPTRで指定）をサーチして、そのファイル情報をFILBUFで指定したバッファに書き込みます。

NAMEPTRには、ワイルドカードが使用できます。

ATRの各ビットの意味は次の通りです。

2つ以上のビットをセットした場合には、そのビットのいずれかを検索したら終了します。

	D15 ～ D08	D07	D06	D05	D04	D03	D02	D01	D00
ATR	0	0	0	A	D	V	S	H	R

- R：1 ＝ 読み込み専用
- H：1 ＝ 隠しファイル
- S：1 ＝ システムファイル
- V：1 ＝ ボリューム名
- D：1 ＝ ディレクトリ名
- A：1 ＝ 通常のファイル（アーカイブ）

スタックに10バイト使用するため、ADDQ. LではなくLEAで戻してください。

nfiles

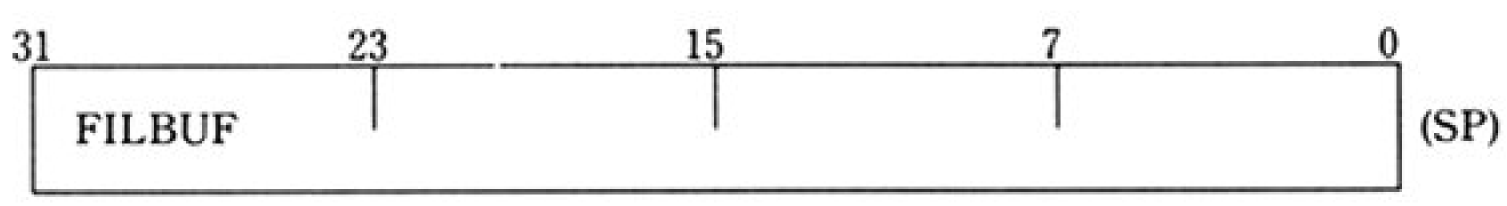
● nfiles (FILBUF)　　　DOS コール番号：$FF4F

機　　能　次のファイルを検索し、ファイル情報を得ます。

引　　数　FILBUF (w)　address　　バッファ（ファイル情報）へのポインタ

コ ー ル

```
         PEA      FILBUF
         DC. W    _NFILES
         ADDQ. L  # 4, SP
           ⋮
FILBUF:  DC. B    ATR          *  ┐
         DC. B    DRIVENO      *  │
         DC. W    DIRCLS       *  │ Human68k 内部で使用
         DC. W    DIRFAT       *  ├ 壊すと nfiles($FF4F)でき
         DC. W    DIRSEC       *  │ なくなる
         DC. W    DIRPOS       *  │
         DC. B    'FILENAME'   *  ┘
         DC. B    'EXT'        *
         DS. B    1            *アトリビュートの一致
         DS. W    1            *ファイルの変更最終時刻
         DS. W    1            *ファイルの変更最終月日
         DS. L    1            *ファイルサイズ
         DS. B    23           *ファイル名(主ファイル名+'.'
                                +拡張子+0)
                               *計 53 バイト
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　　説　files ($FF4E)で求めた FILBUF を使って、次のファイルを検索し、そのファイル情報を再度 FILBUF に書き込みます。

setpdb

● setpdb (PDBADR)　　DOS コール番号：$FF50

機　　能	管理プロセスを移します。
引　　数	PDBADR (r)　address　プロセス

コール

```
        PEA       PDBADR
        DC.W      _SETPDB
        ADDQ.L    #4,SP
```

31	23	15	7	0	
PDBADR					(SP)

リターン	D0. L　　前の PDBADR

解　　説

PDBADR で指定したプロセスに管理を移します。
PDBADR は、プログラムの先頭アドレス－$F0 です。
PDBADR は、getpdb ($FF51)で得られたアドレスでなければいけません。
リターンの D0. L は、前の PDBADR です。

getpdb

● getpdb ( )　　DOS コール番号：$FF51

機　　能	現在のプロセス情報を得ます。
引　　数	なし
コ ー ル	DC. W　　_GETPDB
リターン	D0. L　　現在の PDBADR
解　　説	現在のプロセスを表す PDBADR を得ます。 PDBADR は、プログラムの先頭アドレス－$F0 です。 リターンの D0. L は、現在の PDBADR です。

setenv

● setenv (SETNAME, ENV, SETLINE)　　　DOS コール番号：$FF52

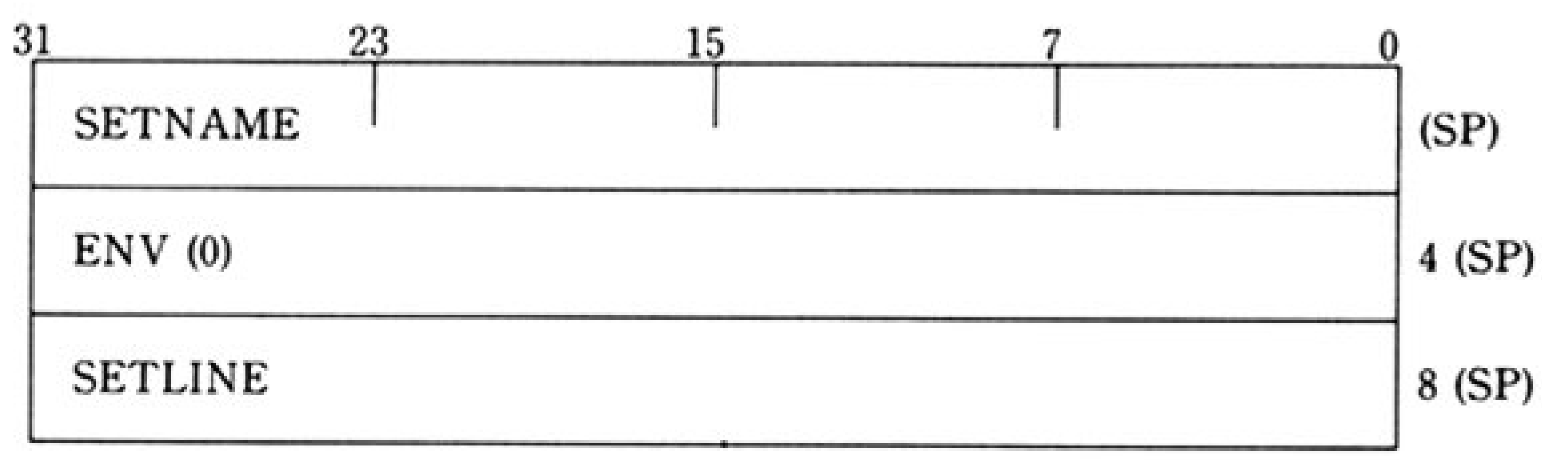
機　能	環境変数を設定します。

引　数

SETNAME (r)	address	文字列（環境変数名）のポインタ
ENV (r)	address	環境ポインタ
SETLINE (r)	address	文字列（変数の値）のポインタ

コール

```
            PEA       SETLINE
            CLR. L    -(SP)
            PEA       SETNAME
            DC. W     _SETENV
            LEA       12(SP), SP
              ⋮
SETNAME: DC. B        'file', 0
SETLINE: DC. B        'TEMP. TXT', 0
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　説　SETNAME で指定した環境変数に、SETLINE で指定した変数値を設定します。

ENV の環境ポインタで 0 を指定すると、自分の環境に設定します（上記コールの例）。

SETLINE に 0 を指定すると、SETNAME の環境変数を消去します。

環境変数の内容は 255 バイトまでです。

getenv

● getenv (GETNAME, ENV, GETBUF)　　　DOS コール番号：$FF53

機　能　環境変数の内容を得ます。

引　数

GETNAME (r)	address	文字列（環境変数名）のポインタ
ENV (r)	address	環境ポインタ
GETBUF (w)	address	バッファのポインタ

コール

```
            PEA     GETBUF
            CLR. L  -(SP)
            PEA     GETNAME
            DC. W   _GETENV
            LEA     12(SP), SP
              ⋮
GETNAME:    DC. B   'file', 0
GETBUF:     DS. B   256
```

31　　　23　　　15　　　7　　　0	
GETNAME	(SP)
ENV (0)	4 (SP)
GETBUF	8 (SP)

リターン　D0. L

D0. L ＜ 0……エラーコード

解　説　ENV で指定した環境から、GETNAME で指定した変数の内容をとり出し、GETBUF で指定したバッファに書き込みます。

ENV の環境ポインタで 0 を指定すると、自分の環境を指定したことになります（上記コールの例）。

バッファは 256 バイトです。

コールのように呼び出すと、バッファには次の内容が書き込まれます。

```
GETBUF :DC. B    'TEMP. TXT', 0
        DS. B    247     *バッファのあまり
```

verifyg

● verifyg ( )　　　DOS コール番号：$FF54

機　能	ベリファイフラグの設定状況を調べます。
引　数	なし
コール	DC. W　_VERIFYG
リターン	D0. L　　設定状況
解　説	ディスクなどに書き込み時のベリファイフラグの設定状況を調べます。 リターンの D0. L は設定状況です。 内容を次に示します。 D0. L ＝ 0……ベリファイしない D0. L ＝ 1……ベリファイする

common

● common (MD, NAME [, POS] [, BUFFER] [, ID_PSP] [, LEN])
DOS コール番号：$FF55

機　　能　common 領域の制御を行います。

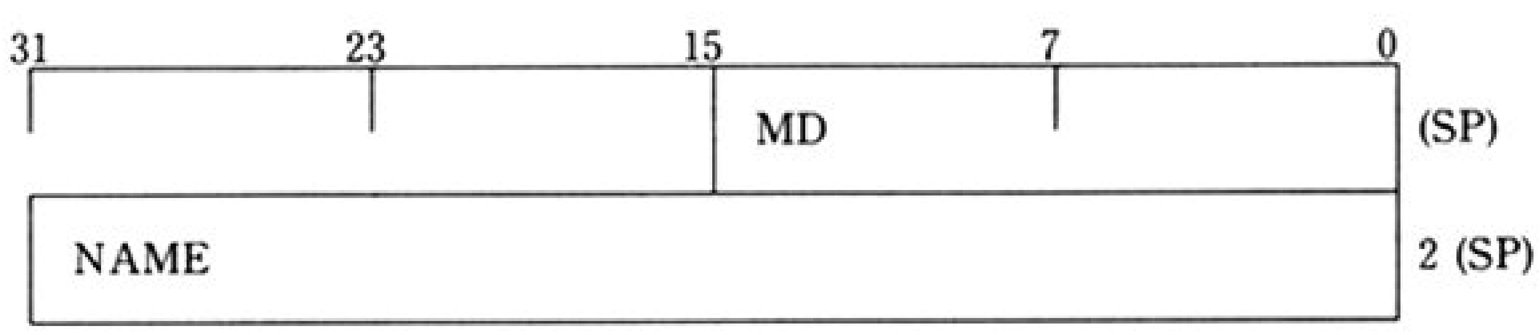
引　　数

MD (r)	word	コントロールモード（この値により引数が異なります）
NAME (r)	address	common 領域名
POS (r)	long	ポジション（common 領域の先頭からのバイト数）
BUFFER (w)	address	バッファ
ID_PSP (r)	long	プロセス ID
LEN (r)	long	バイト数

コ　ー　ル

〈MD ＝ 0, 5〉

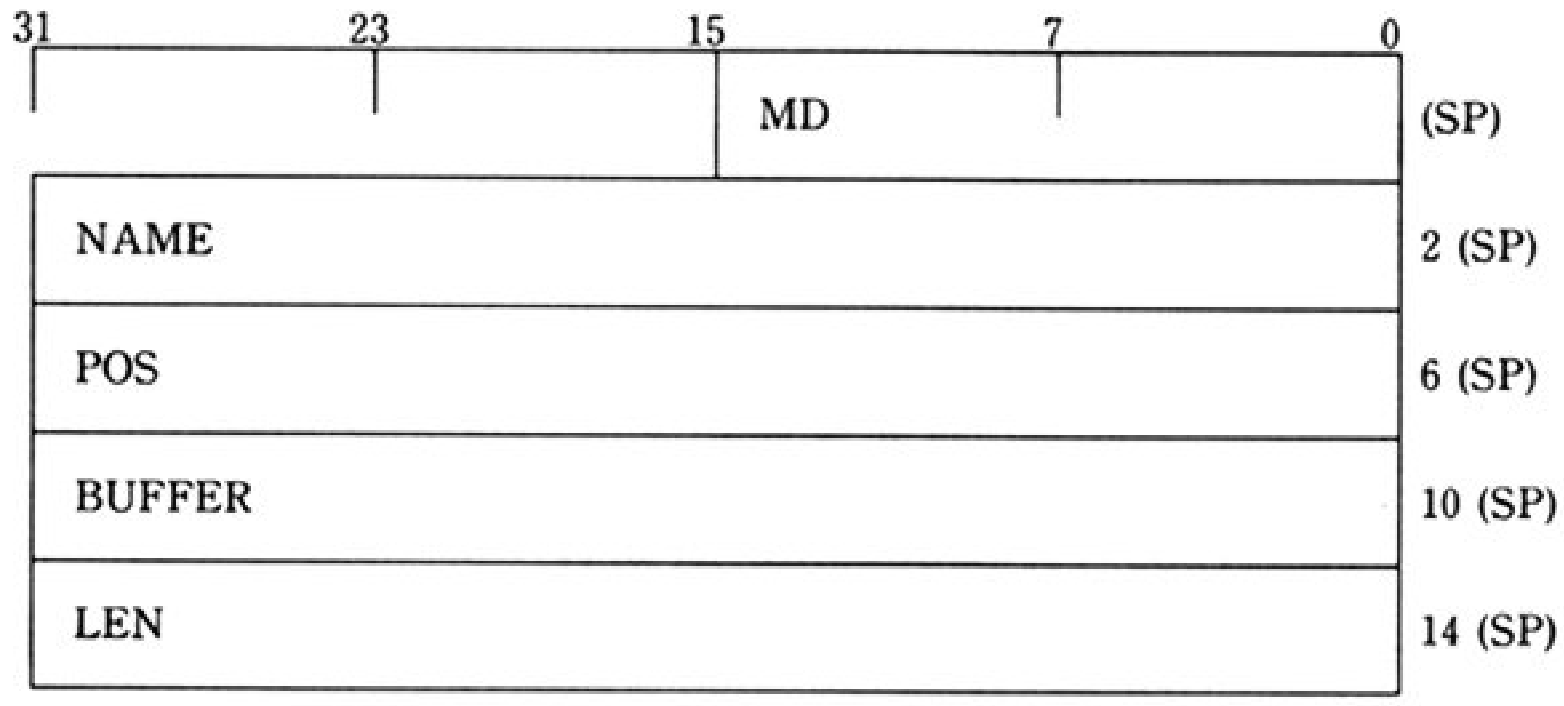
```
        PEA         NAME
        MOVE.W      MD, -(SP)
        DC. W       _COMMON
        ADDQ. L     #6, SP
          ⋮
NAME:   DC. B       'COMAREA', 0
```



〈MD ＝ 1, 2〉

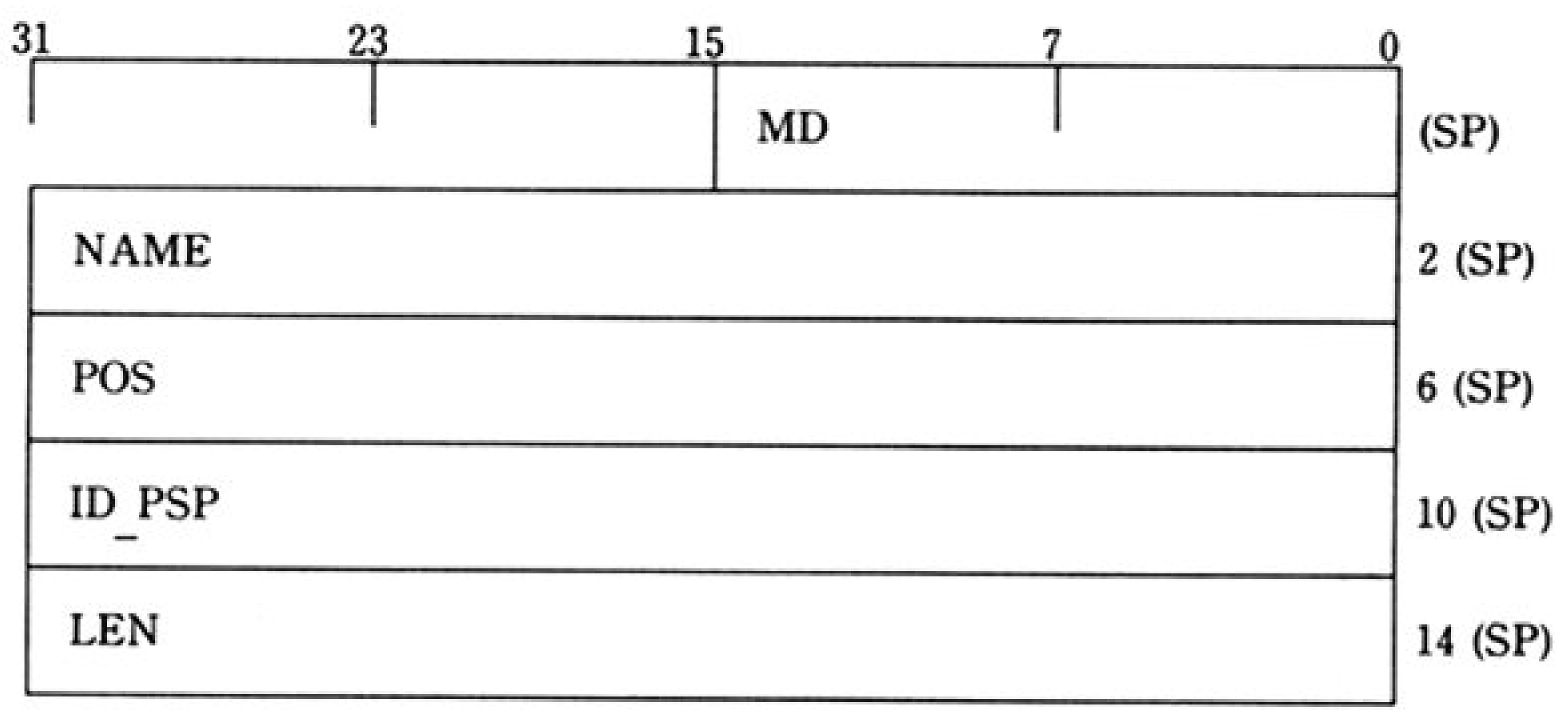
```
        MOVE. L     LEN, -(SP)
        PEA         BUFFER
        MOVE. L     POS, -(SP)
        PEA         NAME
        MOVE. W     MD, -(SP)
        DC. W       _COMMON
        LEA         18 (SP), SP
          ⋮
NAME:   DC. B       'COMAREA', 0
```

```
BUFFER:  DS. B          1024            *サイズは任意
```



〈MD = 3, 4〉

```
         MOVE. L     LEN, -(SP)
         MOVE. L     ID_PSP, -(SP)
         MOVE. L     POS, -(SP)
         PEA         NAME
         MOVE. W     MD, -(SP)
         DC. W       _COMMON
         LEA         18 (SP), SP
            ⋮
NAME:    DC. B       'COMAREA', 0
```



リターン

〈MD = 0〉

D0. L　　common 領域のバイト数

D0. L ＜ 0……エラーコード

〈MD = 1, 2〉

D0. L　　実際に読み書きしたバイト数

D0. L ＜ 0……エラーコード

〈MD ＝ 3～5〉

D0. L　　　エラーコード

解　　説　common 領域の制御を行います。

MDの値によって動作が異なります。

MD ＝ 0 …… NAMEで指定されたcommon領域があるかどうかチェックします。
存在した場合は、その領域のバイト数をD0. L返します。
存在しなかった場合は、エラーコードをD0. Lに返します。

MD ＝ 1 …… NAMEで指定されたcommon領域の、POSからBUFFERへLENバイトだけ読み込みます。
読み込んだバイト数か、エラーコード（負の数）をD0. Lに返します。

MD ＝ 2 …… NAMEで指定されたcommon領域の、POSへBUFFERからLENバイトだけ書き込みます。
NAMEで指定したcommon領域が存在しない場合は、新しく作成し、BUFFERよりデータを書き込みます。
LEN ＝ 0で切り詰めます。
書き込んだバイト数か、エラーコード（負の数）をD0. Lに返します。

MD ＝ 3 …… NAMEで指定したcommon領域の、POSからLENバイトだけ、ID_PSPをつけてロックします。
ID_PSPで指定されたプロセス以外からのアクセスを禁止します。
エラーコードをD0. Lに返します。

MD ＝ 4 …… NAMEで指定したcommon領域の、POSからLENバイトだけ、ID_PSPをつけてロック解除します。
ID_PSPで指定されたプロセス以外からのアクセスを許可します。
エラーコードをD0. Lに返します。

MD ＝ 5 …… NAMEで指定されたcommon領域を削除します。
エラーコードをD0. Lに返します。

このDOSコールにより、プロセス間通信が簡単に実現できます。

rename

● rename (OLD, NEW)　　DOS コール番号：$FF56

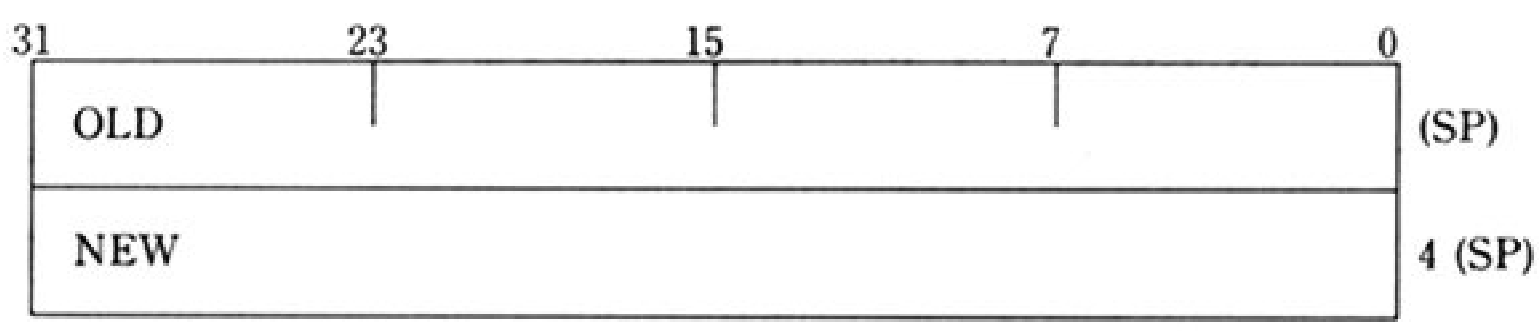
機　能　ファイル名を変更（移動）します。

引　数

OLD (r)	address	文字列（ファイル名）のポインタ
NEW (r)	address	文字列（新しいファイル名）のポインタ

コール

```
        PEA     NEW
        PEA     OLD
        DC. W   _RENAME
        ADDQ. L #8, SP
          ⋮
NEW:    DC. B   'TEST. DOC', 0
OLD:    DC. B   'TEST. BAK', 0
```



リターン　D0. L

D0. L ＜ 0……エラーコード

解　説　OLDで指定したファイル名を、NEWで指定したファイル名に変更します。
OLDとNEWのパス指定が異なる場合は、ファイルの移動になります。
異なるドライブ間のファイル移動はできません。
ただし、OLDまたはNEWがディレクトリ名の場合は、名前の変更はできますが移動はできません。

filedate

● filedate (FILENO, DATETIME)　　DOS コール番号：$FF57

機　　能　ファイルの日付と時間を読み出し／設定します。

引　　数

FILENO (r)	word	ファイルハンドル
DATETIME (r, w)	long	日付と時間

コ ー ル

```
MOVE. L    DATETIME, -(SP)
MOVE. W    FILENO, -(SP)
DC. W      _FILEDATE
ADDQ. L    #6, SP
```

31–16	15–0	
	FILENO	(SP)
DATETIME		2 (SP)

リターン　D0. L　　DATETIME ＝ 0 の場合、ファイルの日時

上位 1 ワード（16 ビット）が$FFFF ならエラーコード（このコールに限って、負の数でもエラーコードでない場合があります）

解　　説　FILENO で指定するファイルハンドルのファイルの、日付と時間を読み出しまたは設定します。

DATETIME ＝ 0 ならファイルの日付と時間を読み出すだけです。

DATETIME ≠ 0 なら、設定になります。

DATETIME およびリターン値のフォーマットは次の通りです。

	D31–D25	D24–D21	D20–D16
DATETIME	年－1980（0～119）	月（1～12）	日（1～31 日）
	D15–D11	**D10–D05**	**D04–D00**
	時（0～23 時）	分（0～59 分）	秒／2（0～29）

秒は 2 秒単位で表し、実際の秒を 2 で割った値になります。

リターンのD0. Lは、DATETIME ＝ 0の場合、読み出したファイルの日時です。

また上位 16 ビットが$FFFF ならエラーコードです。

単に負の数ならばエラーコードということではないので注意してください。

malloc2

● malloc2 (MD, LEN)　　DOS コール番号：$FF58

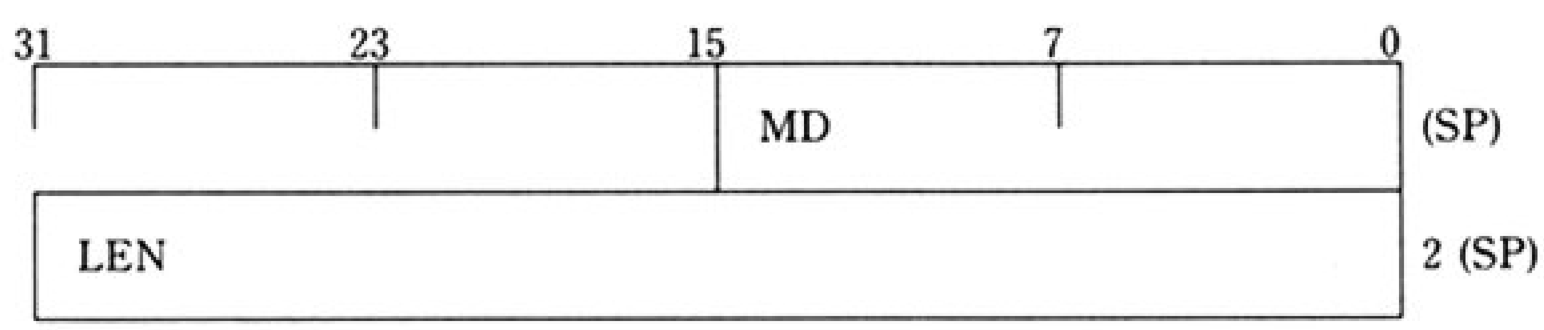
機　　能	メモリを指定された方法で確保します。

引　　数

MD (r)	word	コントロールモード
LEN (r)	long	バイト数

コ ー ル

```
        MOVE. L     LEN, −(SP)
        MOVE. W     MD, −(SP)
        DC. W       _MALLOC2
        ADDQ. L     #6, SP
```



リターン　D0. L　　確保したメモリブロックのポインタ

D0. L ＝ $81??????……確保できない（$??????は最大バイト数）

D0. L ＝ $8200000?……完全に確保不能（?は不定）

解　　説　LEN で指定したバイト数のメモリを確保して、そのポインタを返します。

LEN は、下位 24 ビットのみ有効です。

MD には、メモリを確保する方法を決定するモードを指定します。

MD ＝ 0 ……下位から探し、割り当てる。

MD ＝ 1 ……必要最小ブロックから割り当てる。

MD ＝ 2 ……上位から探し、割り当てる。

リターンの D0. L は、確保されたバイト数のメモリブロックが確保できない場合は$81000000＋最大バイト数を返します。

完全に確保不能の場合は$8200000?を返します。

maketmp

● maketmp (NAMEPTR, ATR)　　　DOS コール番号：$FF5A

機　　能　テンポラリファイルを作成します。

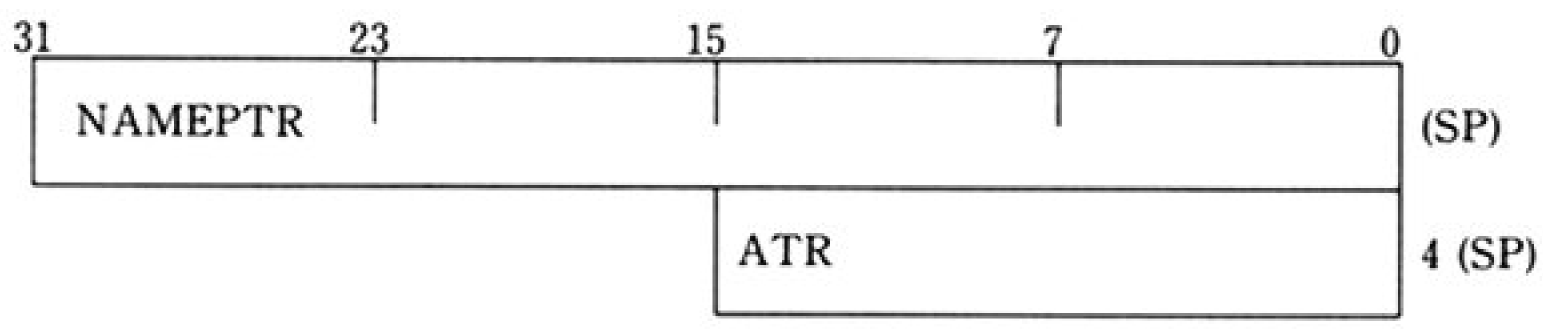
引　　数

NAMEPTR address (w)　　文字列（ファイル名）へのポインタ

ATR (r)　　word　　ファイル属性

コ ー ル

```
          MOVE. W     ATR, -(SP)
          PEA         NAMEPTR
          DC. W       _MAKETMP
          ADDQ. L     #6, SP
             :
NAMEPTR : DC. B       'A : ¥SAMPLE¥TMP???.TXT'
```



リターン　D0. L　　　作成したファイルのファイルハンドル

D0. L ＜ 0……エラーコード

解　　説　NAMEPTR で指定されたパスに、テンポラリファイルを作成します。

NAMEPTR の指すパス名は、次のように設定します。

A : ¥SAMPLE¥FILE????. TXT

"????'の部分が数字に置き換えられ、"0000"～"9999"と変化します。"????"の代わりに数字を最初から設定しておくと、その数字より検索を始めます。

NAMEPTR の指すパス名は、このコールにより書き換えられることに注意してください。

ATR は、作成するテンポラリファイルのファイル属性を指定します。

ATR の各ビットは以下の通りです。

	D15 ~ D08	D07	D06	D05	D04	D03	D02	D01	D00
ATR	0	0	0	A	D	V	S	H	R

- R: 1 = 読み込み専用
- H: 1 = 隠しファイル
- S: 1 = システムファイル
- V: 1 = ボリューム名
- D: 1 = ディレクトリ名
- A: 1 = 通常のファイル（アーカイブ）

テンポラリファイルが正常に作成されたならば、そのファイルハンドルをリターン値として D0. L に返します。
負の数ならエラーコードです。

newfile

● newfile (NAMEPTR, ATR)　　　DOSコール番号：$FF5B

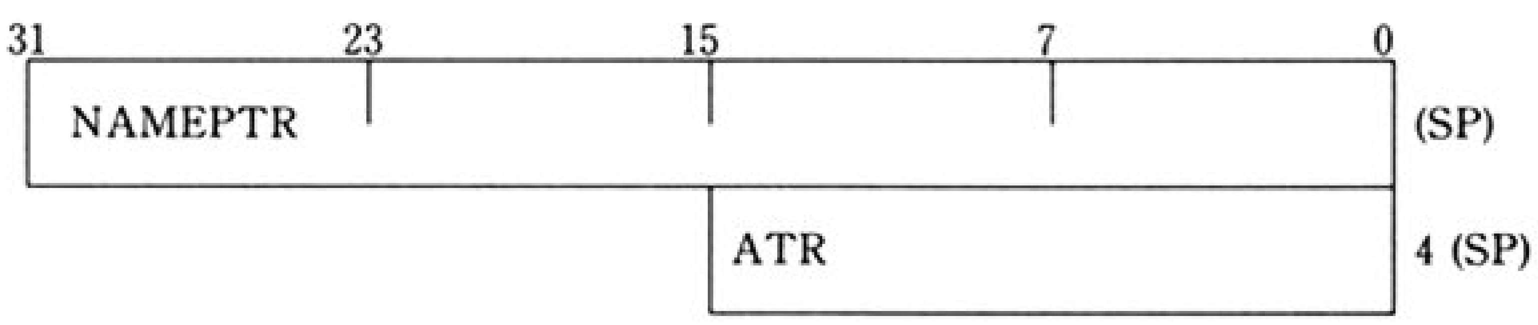
機　　能　ファイルを作成します。

引　　数

NAMEPTR (r)	address	文字列（ファイル名）のポインタ
ATR (r)	word	ファイル属性

コ ー ル

```
            MOVE. W    ATR, -(SP)
            PEA        NAMEPTR
            DC. W      _NEWFILE
            ADDQ. L    #6, SP
              ⋮
NAMEPTR:    DC. B      'A:¥DOC¥TEST. DOC', 0
```



リターン　D0. L　　　作成したファイルのファイルハンドル

D0. L ＜ 0……… エラーコード

D0. L ＝ −80……指定したファイルがすでに存在している

解　　説　新しいファイルを作成します。

NAMEPTRで指定したファイル名のファイルがすでに存在している場合、エラー(−80)を返します。

ATRには、作成するファイルの属性を指定します。

各ビットの内容は次の通りです。

	D15 ~ D08	D07	D06	D05	D04	D03	D02	D01	D00
ATR	0	0	0	A	D	V	S	H	R

- R: 1 = 読み込み専用
- H: 1 = 隠しファイル
- S: 1 = システムファイル
- V: 1 = ボリューム名
- D: 1 = ディレクトリ名
- A: 1 = 通常のファイル（アーカイブ）

ファイルが正常に作成できたら、D0. L に作成したファイルのファイルハンドルを返します。

負の数ならばエラーコードです。

newfile と create($FF3C)の違いは、create($FF3C)の場合、指定したファイル名のファイルがすでに存在していたら前のファイルの内容が捨てられるのに対し、newfile の場合は、ファイル作成を中止してエラーを返します。

lock

● lock (MD, FILENO, OFFSET, LEN)　　　DOSコール番号：$FF5C

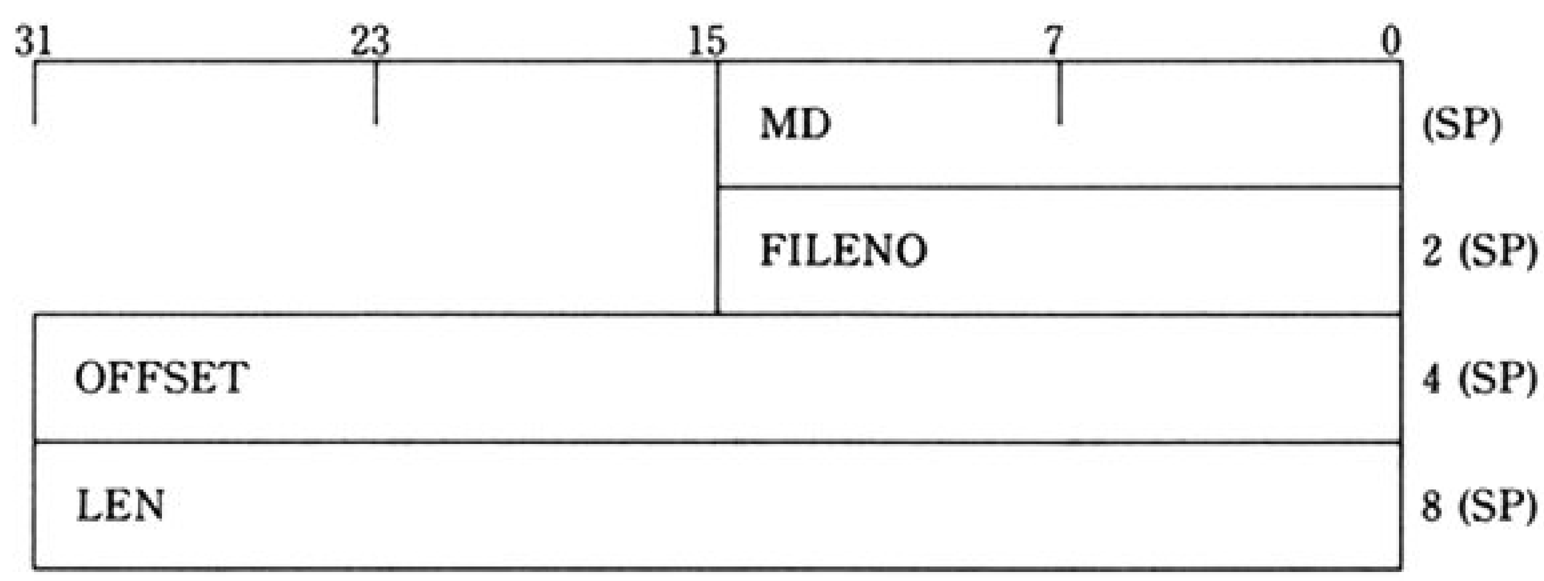
機　能　ファイルのロックを行います。

引　数

MD (r)	word	モード
FILENO (r)	word	ファイルハンドル
OFFSET (r)	long	ファイルの先頭からのオフセット
LEN (r)	long	バイト数

コール

```
MOVE. L    LEN, −(SP)
MOVE. L    OFFSET, −(SP)
MOVE. W    FILENO, −(SP)
MOVE. W    MD, −(SP)
DC. W      _LOCK
LEA        12 (SP), 12
```



リターン　D0. L　　エラーコード

解　説　指定したファイルに対するアクセスのロック／ロック解除を行います。

ファイルのロックとは、ファイルアクセスの排他制御であり、他のプロセスからのファイルアクセスを禁止することです。

MD＝0のときファイルをロックします。

　ロックする領域は、ファイルの先頭＋OFFSETから、LENバイト分だけです。

MD＝1のときファイルをロック解除します。

　ロック解除する領域は、ファイルの先頭＋OFFSETから、LENバイト分だけです。

リターンのD0. Lはエラーコードです。

assign

● assign (MD, BUFF1 [, BUFF2] [, MODE])　　DOS コール番号：$FF5F

機　能　仮想ドライブ・仮想ディレクトリの割り当てリストの取得／作成／キャンセルを行う。

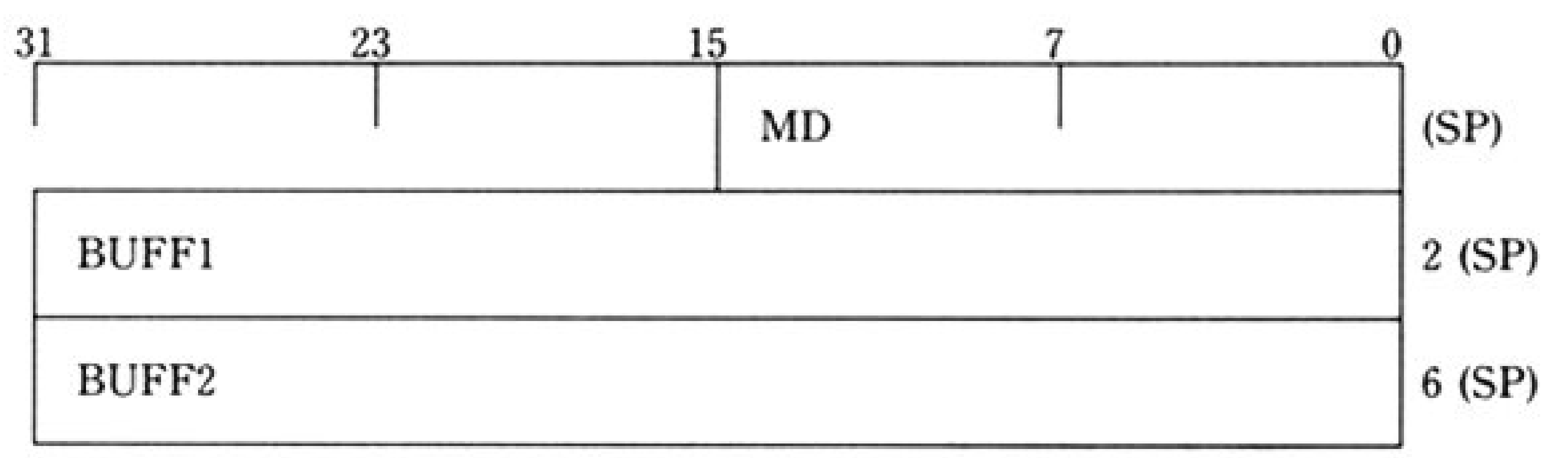
引　数

MD (r)	word	動作モード
BUFF1 (r)	address	仮想ドライブ・仮想ディレクトリへのポインタ
BUFF2 (w)	address	実ドライブ・実ディレクトリへのポインタ
MODE (r)	word	割り当てモード

コール

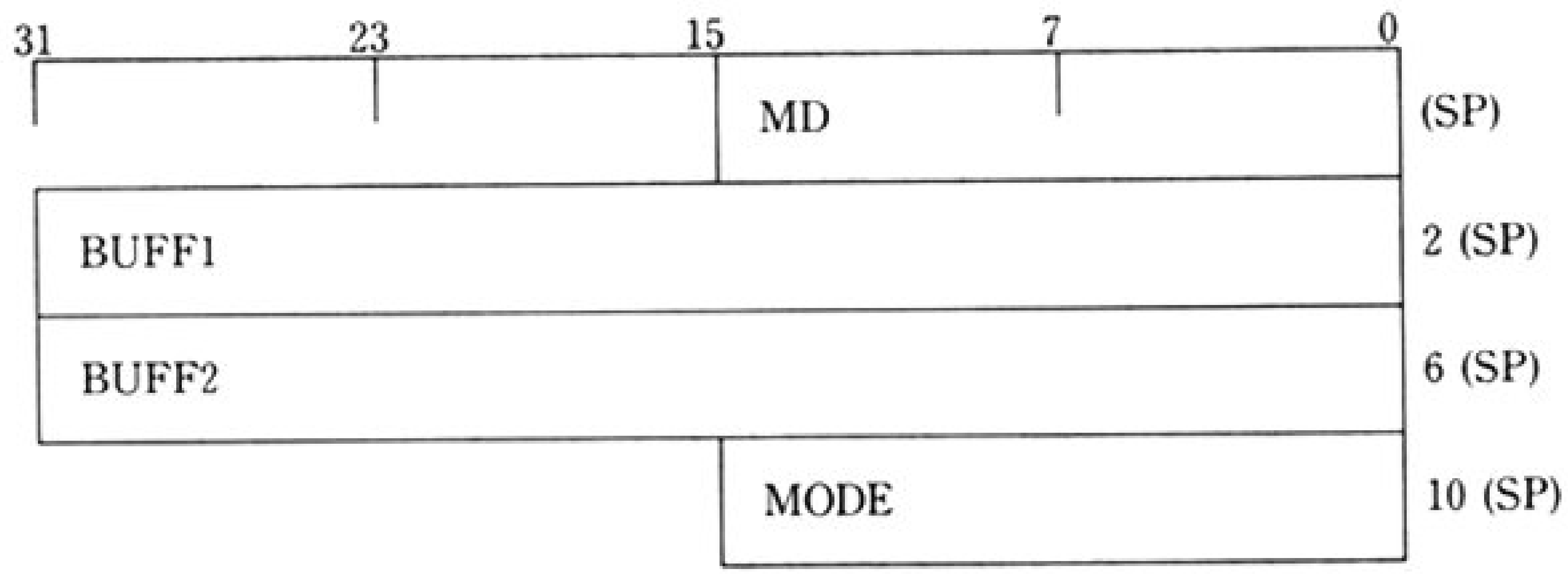
〈MD = 0〉

```
         PEA       BUFF2
         PEA       BUFF1
         MOVE. W   MD, -(SP)
         DC. W     _ASSIGN
         LEA       10 (SP), SP
           ⋮
BUFF1:   DC. B     'A : ¥LIB', 0
BUFF2:   DS. B     256
```



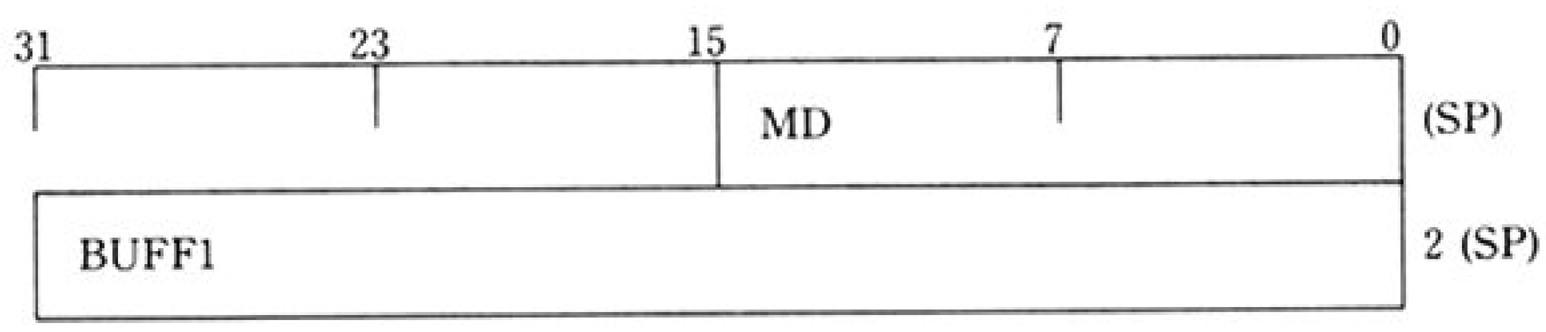
〈MD = 1〉

```
         MOVE. W   MODE, -(SP)
         PEA       BUFF2
         PEA       BUFF1
         MOVE. W   MD, -(SP)
         DC. W     _ASSIGN
         LEA       12 (SP), SP
           ⋮
BUFF1:   DC. B     'A : ¥LIB', 0
BUFF2:   DC. B     'Z :'
```



〈MD ＝ 4〉

```
            PEA         BUFF1
            MOVE. W     MD, −(SP)
            DC. W       _ASSIGN
            ADDQ, L     #6, SP
              ⋮
BUFF1:      DC. B       'A : ¥LIB', 0
```



リターン

〈MD ＝ 0〉

D0. L　　　割り当てモード

D0. L ＝ $40……割り当てが存在しない

D0. L ＝ $50……仮想ドライブの割り当て

D0. L ＝ $80……仮想ディレクトリの割り当て

D0. L ＜ 0………エラーコード

〈MD ＝ 1, 4〉

D0. L

D0. L ＜ 0………エラーコード

解　説

指定された仮想ドライブ・仮想ディレクトリの割り当てリストの、取得／作成／キャンセルを行います。

モードMDの値により動作が異なります。

MD ＝ 0……割り当てリストの取得を行います。

BUFF1には、ドライブ名へのポインタを指定します。

BUFF2には、ディレクトリ名を格納する領域へのポインタを指定します。
リターン値として、以下に示す割り当てモードを返します。
なお、負の数ならエラーコードです。

・戻り値が0x40のとき（割り当てが存在しない）
BUFF1で指定されたドライブのカレントディレクトリがBUFF2へセットされます。
・戻り値が0x50のとき（仮想ドライブの割り当て）
BUFF1をアクセスしたときに実際にアクセスされるディレクトリがBUFF2へセットされます。
・戻り値が0x60のとき（仮想ディレクトリの割り当て）
BUFF1を実際にアクセスしようとしたときにアクセスしなければならないディレクトリがBUFF2にセットされます。

MD ＝ 1……割り当てリストの作成を行います。
BUFF1には、ドライブ名へのポインタを指定します。
BUFF2には、割り当てる実ディレクトリ名へのポインタを指定します。
MODEには、以下に示す割り当てモードを指定します。

・MODEが0x50のとき（仮想ドライブの割り当て）
BUFF1のドライブをアクセスしようとするとBUFF2のディレクトリをアクセスする。
・MODEが0x60のとき（仮想ディレクトリの割り当て）
BUFF2のディレクトリをアクセスすると、BUFF1のドライブをアクセスする。

戻り値は負の数ならエラーコードです。
MD ＝ 4……割り当てリストのキャンセルを行います。
BUFF1には、ドライブ名へのポインタを指定します。
戻り値は負の数ならエラーコードです。

s_malloc

● s_malloc (MD, LEN)　　　DOS コール番号：$FF7D

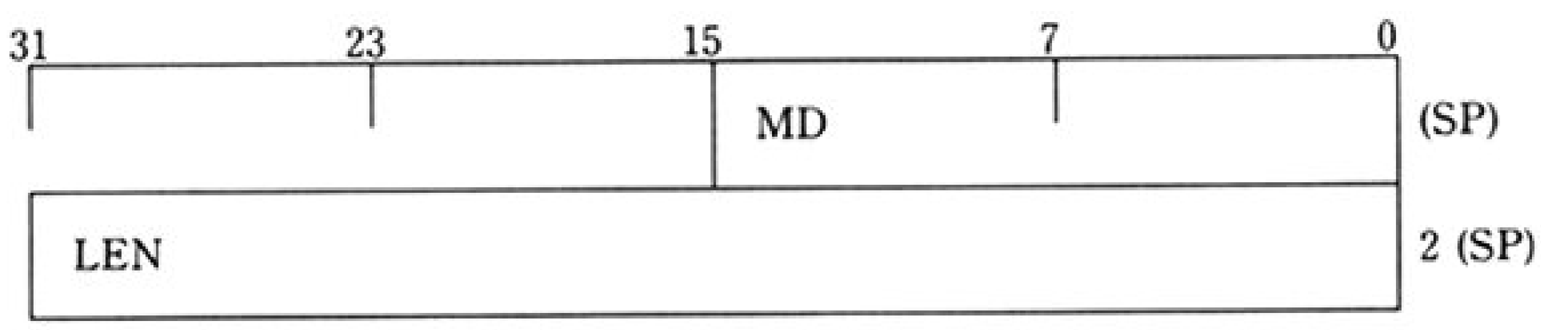
機　　能　メインのメモリ管理下からメモリブロックを確保します。

引　　数

MD (r)	word	コントロールモード
LEN (r)	longs	バイト数

コ ー ル

```
        MOVE. L     LEN, -(SP)
        MOVE. W     MD, -(SP)
        DC. W       _S_MALLOC
        ADDQ. L     #6, SP
```



リターン　D0. L　　確保したメモリブロックのポインタ

D0. L ＝ $81?????? ……確保できない（$??????は最大バイト数）

D0. L ＝ $8200000? …完全に確保不可能（?は不定）

解　　説　LEN で指定したバイト数のメモリを、メインのメモリ管理下から確保し、そのポインタを返します。

LEN は、下位 24 ビットのみ有効です。

MD には、メモリを確保する方法を決定するモードを指定します。

MD ＝ 0……下位から探し、割り当てる

MD ＝ 1……必要最小ブロックから割り当てる

MD ＝ 2……上位から探し、割り当てる

リターンの D0. L は、確保したメモリブロックのポインタ（メモリ管理ポインタは－$10）を返します。

指定されたバイト数のメモリブロックが確保できない場合は$81000000＋最大バイト数を返します。

完全に確保不能の場合は$8200000?を返します（?は不定）。

s_malloc は、通常のアプリケーションプログラムで使用してはいけません。

s_malloc を使用するプログラムは、OS から起動されて、終了することができない（常駐プロセスのように exit や exit2 の DOS コールを行なわない）プログラムでなければなりません。

s_mfree

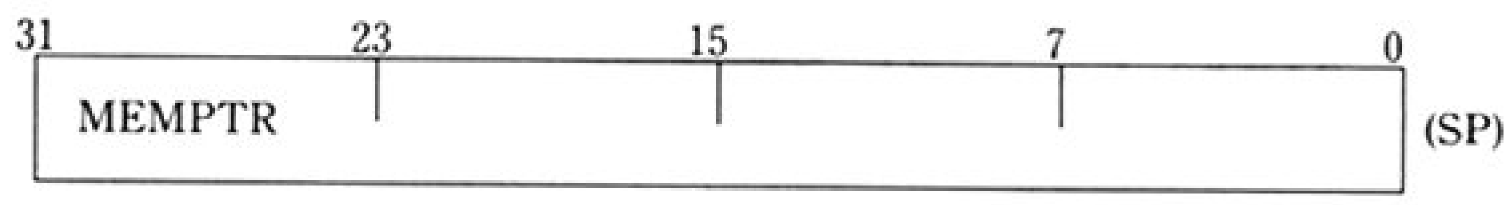
● s_mfree (MEMPTR)　　DOSコール番号：$FF7E

機　能	メインのメモリの管理下のメモリブロックを開放します。

引　数　MEMPTRT (r)　address　メモリブロックへのポインタ

コール

```
PEA       MEMPTR
DC. W     _S_MFREE
ADDQ. L   #4, SP
```



リターン　D0. L　エラーコード

解　説

MEMPTRで指定した、メインのメモリ管理下のメモリブロックを開放します。
MEMPTRは、malloc（$FF48）やs_malloc（$FF7D）で返されたD0. Lの値（確保したメモリブロックのポインタ）です。
MEMPTRがs_process（$FF7F）で指定したサブのメモリ管理ポインタで、かつそのスレッドIDがカレントIDならば、kill_pr（$FFF9）でプロセスを削除します。
その場合、サブのメモリ管理下で常駐終了しているプロセス（スレッドID）は、メインのメモリ管理下に入ります。
またs_mfreeを行う場合は、指定したMEMPTRのメモリブロック中で動作しているプログラムが存在しない状態でのみ行ってください。
MEMPTR ＝ 0の場合、自分が確保したメモリをすべて開放します。
子プロセスも、子プロセスが確保したメモリもすべて開放します。
リターンのD0. Lはエラーコードです。

s_process

● s_process (ID, START, LENGTH, I_LEN)　　　DOS コール番号：$FF7F

機　　能　サブのメモリ管理を設定します。

引　　数

ID (r)	word	プロセスID（スレッドID）
START (w)	address	サブのメモリ管理ポインタ
LENGTH (r)	long	メモリブロック全体のバイト数
I_LEN (r)	long	先頭ブロックのバイト数

コール

```
MOVE.L    I_LEN, -(SP)
MOVE.L    LENGTH, -(SP)
PEA       START
MOVE.W    ID, -(SP)
DC.W      _S_PROCESS
LEA       14(SP), SP
```

31–16	15–0	
	ID	(SP)
START		2 (SP)
LENGTH		6 (SP)
I_LEN		10 (SP)

リターン　D0.L　　先頭のメモリブロック（サイズはI_LEN）へのポインタ

D0.L ＝ $FFFF00??……IDエラー（??はIDの最大値）

D0.L ＝ −14……………LENGTHがI_LEN＋16より小さい

解　　説　指定した先頭アドレスとバイト数で、サブのメモリ管理を設定します。

これ以後、指定したIDをもつスレッド（プロセス）からのメモリ確保要求は、このサブのメモリ管理下に限定されます。

サブのメモリ管理を新しく設定する場合には、必ず先頭から1ブロックだけ自動的に確保されるので、そのバイト数をI_LENで指定します。

STARTからの16バイトは、メモリ管理ポインタが占有するため破壊されます。

ID ＝ 0はメインスレッドなので変更できません。

diskred

● diskred (ADR ［＋$80000000］, DRIVE, SECTLEN)

DOS コール番号：$FFF3

機　　能　ブロックデバイスから直接読み込みます。

引　　数　〈通常のドライブの場合〉

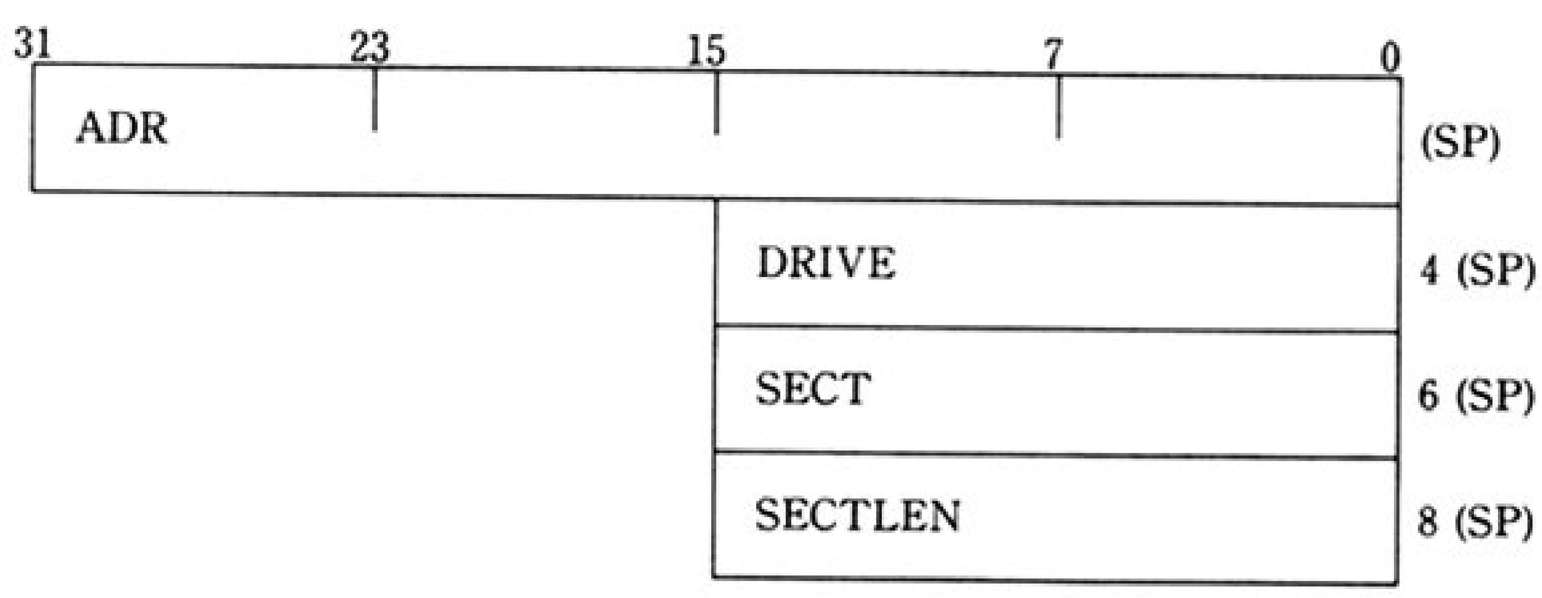
ADR (w)	address	読み込みバッファのアドレス（1024 バイト単位）
DRIVE (r)	word	読み込むドライブ名
SECT (r)	word	読み込み始めるセクタ番号
SECTLEN (r)	word	読み込むセクタ数

〈大容量ドライブ・特殊ドライブの場合〉

ADR+$80000000(w)	address	読み込みバッファのアドレス
DRIVE (r)	word	読み込むドライブ名
SECT (r)	long	読み込み始めるセクタ番号
SECTLEN (r)	long	読み込むセクタ数

コ　ー　ル　〈通常のドライブの場合〉

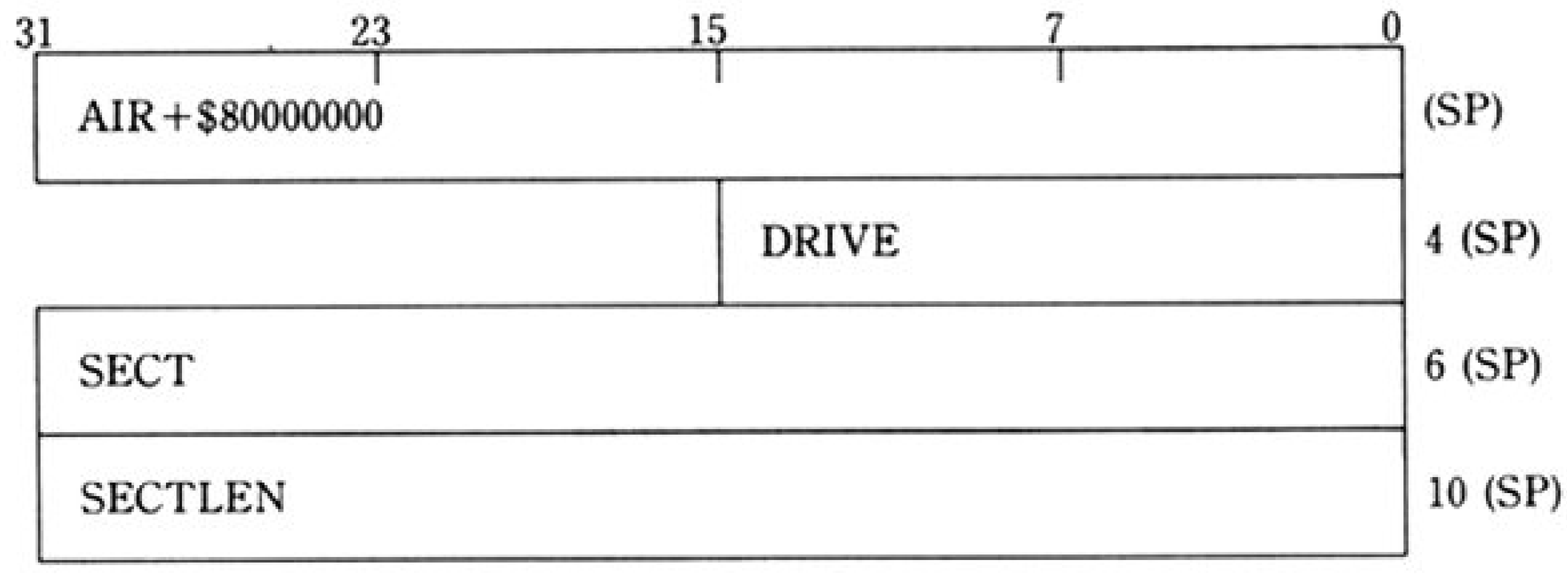
```
MOVE. W    SECTLEN, -(SP)  ＊サイズは word
MOVE. W    SECT, -(SP)     ＊サイズは word
MOVE. W    DRIVE, -(SP)
PEA        ADR
DC. W      _DISKRED
LEA        10 (SP), SP
```



〈大容量ドライブ・特殊ドライブの場合〉

```
MOVE. L    SECTLEN, -(SP)  ＊サイズは long
MOVE. L    SECT, -(SP)     ＊サイズは long
MOVE. W    DRIVE, -(SP)
LEA        ADR, A0
```

```
ADDA. L     #$80000000, A0
MOVE. L     A0, -(SP)          *アドレスは負の数
DC. W       _DISKRED
LEA         14 (SP), SP
```



リターン　なし

解　　説　DRIVEで指定したブロックデバイスのSECTセクタから、SECTLEN分だけ直接読み込み、ADRで指定したバッファに書き込みます。
大容量ドライブや特殊ドライブのために、SECT、SECTLENをlongのサイズで指定できるよう拡張されました。
大容量ドライブや特殊ドライブの場合は、読み込みバッファのアドレスに$80000000を足した値をADRとしてコールしてください。
読み込みはセクタ単位で行うので、ADRからのバッファは1セクタあたりのバイト数（通常ドライブの場合は、1024バイト）単位にしてください。
SECTLENは1～?までです（?はメディアによって異なります）。
SECTは0～?までです（?はメディアによって異なります）。
DRIVEは次のように指定します。

DRIVE ＝ 0：カレントドライブ
DRIVE ＝ 1：Aドライブ
DRIVE ＝ 2：Bドライブ
　⋮　　　　⋮

diskwrt

● diskwrt (ADR [+$80000000], DRIVE, SECTLEN)

DOS コール番号：$FFF4

機　能　ブロックデバイスに直接書き込みます。

引　数　〈通常のドライブの場合〉

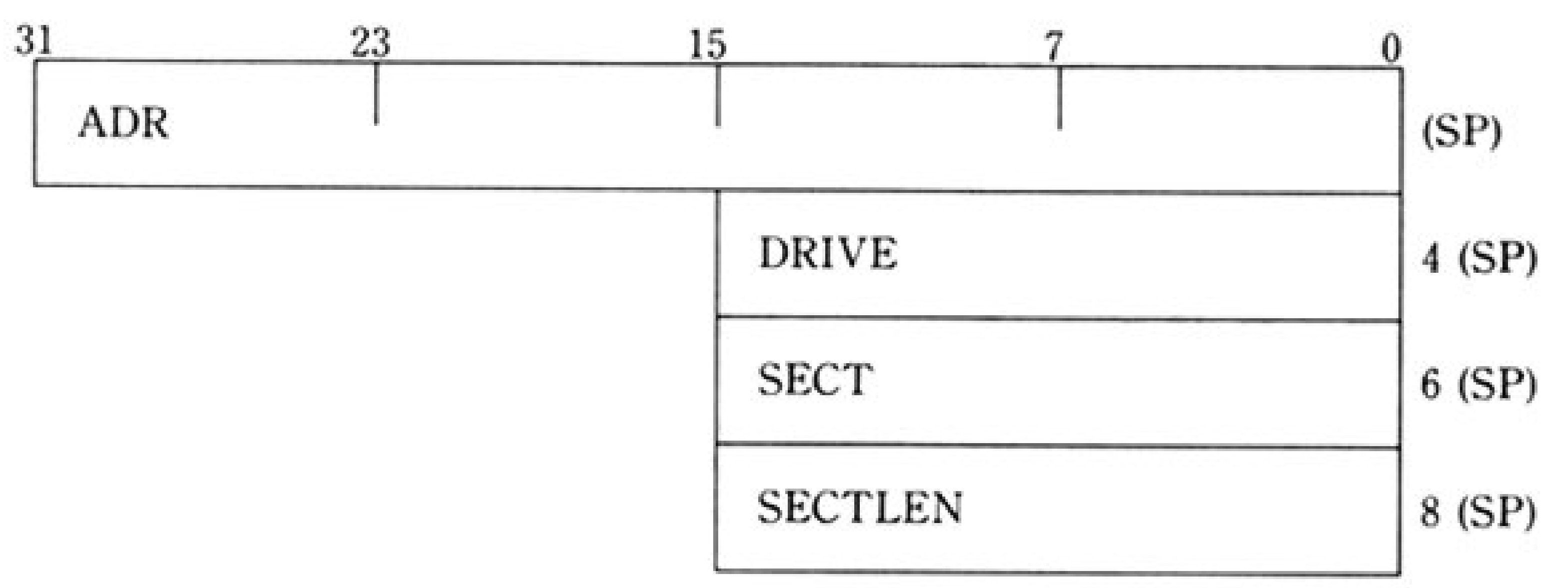
ADR (w)	address	書き込みバッファのアドレス (1024 バイト単位)
DRIVE (r)	word	書き込むドライブ名
SECT (r)	word	書き込み始めるセクタ番号
SECTLEN (r)	word	書き込むセクタ数

〈大容量ドライブ・特殊ドライブの場合〉

ADR+$80000000(w)	address	書き込みバッファのアドレス
DRIVE (r)	word	書き込むドライブ名
SECT (r)	long	書き込み始めるセクタ番号
SECTLEN (r)	long	書き込むセクタ数

コール　〈通常のドライブの場合〉

```
MOVE. W    SECTLEN, -(SP)  *サイズは word
MOVE. W    SECT, -(SP)     *サイズは word
MOVE. W    DRIVE, -(SP)
PEA        ADR
DC. W      _DISKWRT
LEA        10 (SP), SP
```



〈大容量ドライブ・特殊ドライブの場合〉

```
MOVE. L    SECTLEN, -(SP)  *サイズは long
MOVE. L    SECT, -(SP)     *サイズは long
MOVE. W    DRIVE, -(SP)
LEA        ADR, A0
```

```
ADDA. L     #$80000000, A0
MOVE. L     A0, -(SP)        *アドレスは負の数
DC. W       _DISKWRT
LEA         14 (SP), SP
```

31–16	15–0	
ADR+$80000000		(SP)
	DRIVE	4 (SP)
SECT		6 (SP)
SECTLEN		10 (SP)

リターン　なし

解　説　DRIVEで指定したブロックデバイスのSECTセクタからSECTLEN分だけ、ADRで指定したバッファのデータを直接書き込みます。

大容量ドライブや特殊ドライブのために、SECT、SECTLENをlongのサイズで指定できるよう拡張されました。

大容量ドライブや特殊ドライブの場合は、書き込みバッファのアドレスに$80000000を足した値を、ADRとしてコールしてください。

書き込みはセクタ単位で行うので、ADRからのバッファは1セクタあたりのバイト数（通常ドライブの場合は、1024バイト）単位にしてください。

SECTLENは1～?までです（?はメディアによって異なります）。

SECTは0～?までです（?はメディアによって異なります）。

DRIVEは次のように指定します。

DRIVE ＝ 0：カレントドライブ
DRIVE ＝ 1：Aドライブ
DRIVE ＝ 2：Bドライブ
　⋮　　　　⋮

indosflg

● indosflg ( )　　DOS コール番号：$FFF5

機　　能　OS ワーク内へのポインタを返します。

コ ー ル　DC. W　　_INDOSFLG

リターン　D0. L　　INDOS_FLG へのポインタ

解　　説　OS ワーク内へのポインタを返します。

スーパーバイザ領域のため、ユーザーモードではアクセスできませんので注意してください。

ここで返すポインタは、INDOS_FLG を指しています。

INDOS_FLG は、現在 DOS コールを実行中かどうかを示すフラグです。

INDOS_FLG 以後には、OS にとって重要なワークが存在するので、絶対に書き換えないでください。

INDOS_FLG の内容を以下に示します。

indosf :	ds. w	＊OS 実行中のレベル
doscmd :	ds. b	＊OS 実行中ファンクション番号
fat_flg :	ds. b	＊FAT 検索モード（0 ＝ 標準、0 ≠ 先頭から）
retry_count :	ds. w	＊I／O リトライカウント（標準で 3 回）
retry_time :	ds. w	＊I／O リトライ待ち時間（標準で 100（1 秒））
verifyf :	ds. w	＊ベリファイモード（0 ＝ オフ、1 ＝ オン）
breakf :	ds. b	＊ブレークモード（0 ＝ オフ、1 ＝ オン）
ctrlpf :	ds. b	＊CTRL＋P モード（0 ＝ オフ、1 ＝ オン）
	ds. b	＊システムで使用
wkcurdrv :	ds. b	＊カレントドライブ（a：0）

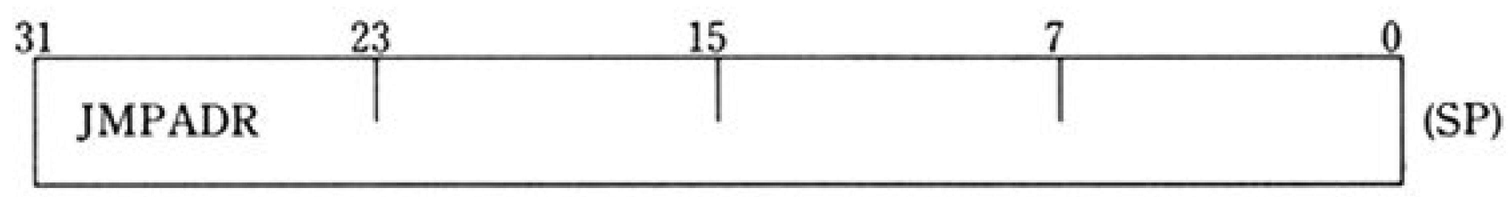
super_jsr

● super_jsr (JMPADR)　　DOS コール番号：$FFF6

機　能	スーパーバイザ領域にあるプログラムをサブルーチンコールします。
引　数	EMPADR (r)　　address　　JSR するアドレス

コール

```
PEA       JMPADR
DC. W     _SUPER_JSR
ADDQ. L   #4, SP
```



リターン　D0-D7／A0-A6　　JSR するプログラムに依存します。

解　説

スーパーバイザ領域にあるプログラムに JSR (Jump to SubRoutine) したい時に使用します。
JMPADR は、JSR したいプログラムのエントリアドレスを指定します。
DOS コールを実行する直前の D0-D7／A0-A6 が、指定したプログラムに渡されます。
また、プログラムから制御が戻るときには、その時の D0-D7／A0-A6 がそのまま返されます。
なお、SR は変化しません。
SSP／USP は、どのように指定したプログラムに渡されるか不定なので、スタックを用いて引数を渡すことはできません。
この DOS コールでは、JSR することにより発生する暴走やバスエラーに対する処理を全く行いません。
この DOS コールのエントリを intvcs ($FF25) で変更することはできません。

bus_err

● bus_err (S_ADR, D_ADR, MD)　　　DOS コール番号：$FFF7

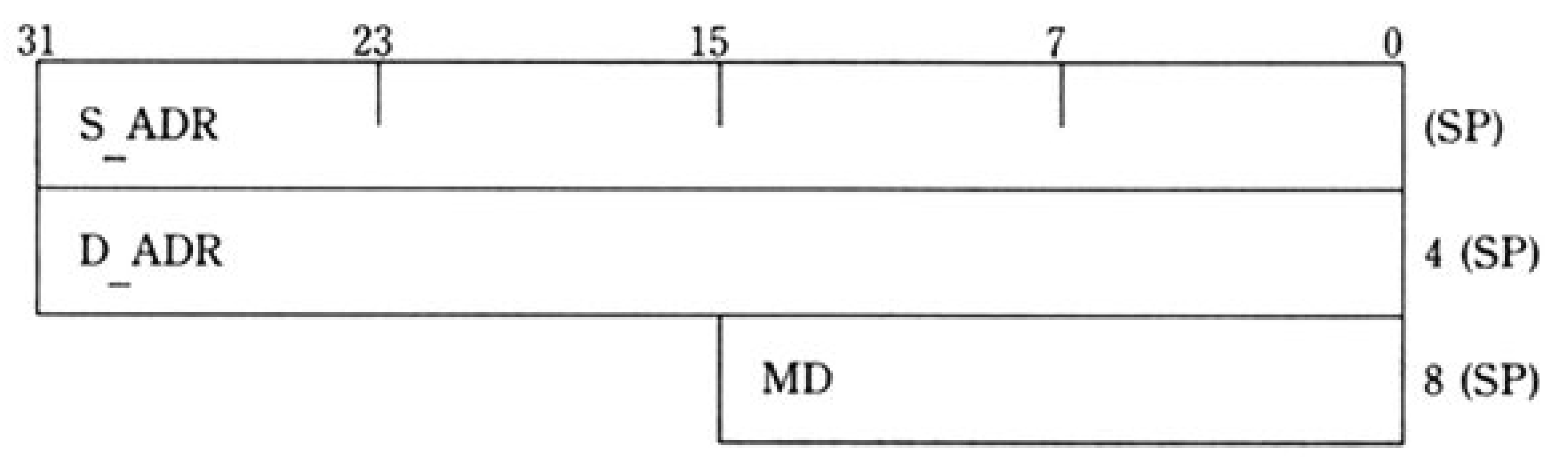
機　　能　バスエラーが発生するかどうかチェックします。

引　　数

S_ADR (r)	address	読み込みポインタ
D_ADR (w)	address	書き込みポインタ
MD (r)	word	アクセスモード

コ ー ル

```
MOVE. W   MD, -(SP)
PEA       D_ADR
PEA       S_ADR
DC. W     _BUS_ERR
LEA       10 (SP), SP
```



リターン　D0. L

D0. L ＝ 0………読み書き可能
D0. L ＝ 1………D_ADR へ書き込んだ時にバスエラーが発生
D0. L ＝ 2………S_ADR から読み込んだ時にバスエラーが発生
D0. L ＝ −1……エラー（引数がおかしい）

解　　説　S_ADR, D_ADR で指定したアドレスに対してそれぞれ読み込み、書き込みを行い、バスエラーが発生するかどうかチェックします。

スーパーバイザ領域や、なにもマッピングされていない領域など、バスエラーが発生するかもしれない領域をアクセスする前に、読み書きできるかどうかをこの DOS コールで事前にチェックすることができます。

モード MD の指定方法は以下の通りです。

MD ＝ 1……バイトで読み書き
MD ＝ 2……ワードで読み書き
MD ＝ 4……ロングで読み書き

リターンは D0. L に、どのようにバスエラーが発生したかを返します。

－1の場合はエラーです。
これは、MDの値が異常な場合や、ワードやロングワードでの読み書きを指定したときにS_ADR, D_ADRに奇数アドレスを指定した場合です。
このコールはintvcs（$FF25）でエントリを変更できません。

open_pr

● open_pr (NAME, COUNT, INIT_USP, INIT_SSP, INIT_SR, INIT_PC, BUFF, SLEEP_TIME)　　DOSコール番号：$FFF8

機能　バックグラウンドタスクを登録します。

引数

NAME (r)	address	スレッド名へのポインタ
COUNT (r)	word	実行間隔を決める値
INIT_USP (r)	address	USP の初期値
INIT_SSP (r)	address	SSP の初期値
INIT_SR (r)	word	SR の初期値
INIT_PC (r)	address	プログラムの実行先頭アドレス
BUFF (w)	address	タスク間通信バッファへのポインタ
SLEEP_TIME (r)	long	待ち時間（ミリ秒単位）

コール

```
        MOVE. L   SLEEP_TIME, -(SP)
        PEA       BUFF
        PEA       INIT_PC
        MOVE. W   INIT_SR, -(SP)
        PEA       INIT_SSP
        PEA       INIT_USP
        MOVE. W   COUNT, -(SP)
        PEA       NAME
        DC. W     _OPEN_PR
        LEA       28 (SP), SP
          ⋮
BUFF:   DC. L     LENGTH        *データ用バッファのバイト数
        DC. L     DATABUF_ADR   *データ用バッファの先頭アド
                                 レス
        DC. W     COMMAND       *コマンド番号
        DC. W     ID            *相手の ID (−1で通信許可)
```

bit 31 … 16	bit 15 … 0	
NAME		(SP)
	COUNT	4 (SP)
INIT_USP		6 (SP)
INIT_SSP		10 (SP)
	INIT_SR	14 (SP)
INIT_PC		16(SP)
BUFF		20 (SP)
SLEEP_TIME		24 (SP)

リターン　D0. L　　登録したタスクのスレッド ID

D0. L ＜ 0……… エラーコード

D0. L ＝ −27……すでに同じ名前のタスクが存在する

D0. L ＝ −29……これ以上タスクを登録できない

解　説　バックグラウンドタスクを登録します。

登録したスレッドは、スリープ状態になります。

NAME には、スレッドの名前を表す 15 文字以内の文字列へのポインタを指定します。

もし、すでに起動しているスレッドと名前が同じ場合は、エラーとなります。

COUNT には、タスクを 1 回実行するのに、タスク切り替えのためのタイマー割り込みの発生を何回カウントするか、その回数を指定します。

COUNT の値は 2〜255 です。

0 または 1 が指定された場合は、2 として扱います。

INIT_USP, INIT_SSP, INIT_PC は、登録したタスクを実行するときの、それぞれのレジスタの初期値です。

システム用のスタックは 6 キロバイト必要です。

また、INIT_SR は$0000 または$2000 を指定し、ユーザーモードかスーパーバイザモードかどちらかを選択します。

上記以外のレジスタの初期値はすべて 0 です。

BUFF には、タスク間通信のバッファへのポインタを指定します。

SLEEP_TIME には、待ち時間をミリ秒単位で指定します。

0 を指定すると、永久にスリープします。

open_pr により、バックグウランドタスクとして登録されたスレッドは、keeppr（$FF31）で常駐終了します。

スレッドをメモリ上から消去したい場合は、kill_pr ($FFF9) を使用します。
リターン値として、登録したバックグラウンドタスクのスレッドIDを返します。
負の数ならエラーコードです。

kill_pr

● kill_pr ( )　　DOS コール番号：$FFF9

機　能	自分自身のプロセスを削除します。
コール	DC. W　_KILL_PR
リターン	D0. L　エラーコード
解　説	自分自身のプロセスを削除します。 常駐終了していたプロセスの場合、同じプロセス ID を持つスレッドを全て削除し、確保されていたメモリを開放します。 kill_pr を実行する時は、自分でオープンしたファイルはすべてクローズし、書き換えたベクタなども元に戻した後にしてください。 システムは、スレッドの削除とメモリの開放しか行いません。 プロセス内で、いくつかの登録したスレッドを実行した後で、exit ($FF00) や exit2 ($FF4C) でプロセスを終了する場合は、プロセス内でオープンしたスレッドのみを kill_pr で削除してから、プロセスを終了してください。 メインのスレッドであるプロセスを kill_pr した場合、その後の動作は保証されません。 自分以外のプロセスを削除したい場合は、send_pr ($FFF0) を用いて KILL コマンドを送ってください。 リターン値はエラーコードです。

get_pr

● get_pr (ID, BUFF)　　DOS コール番号：$FFFA

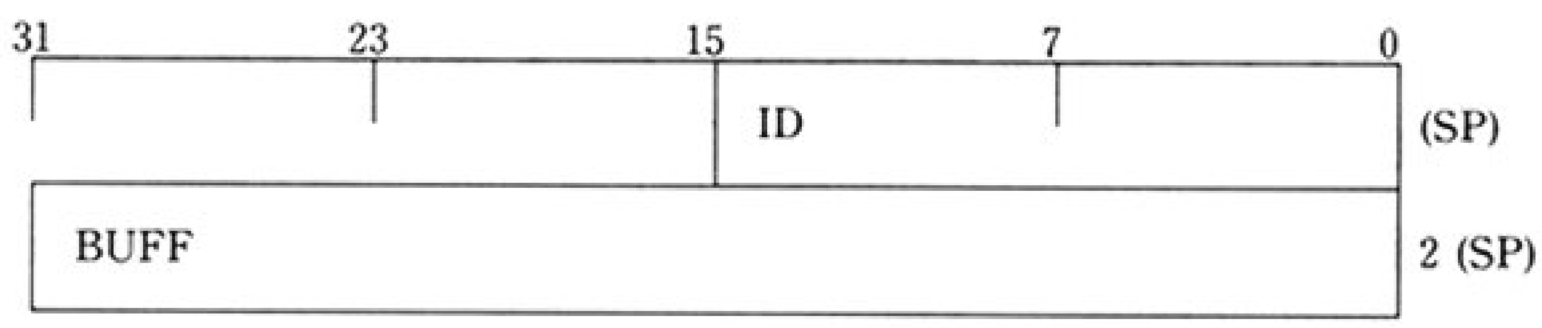
機　　能　スレッドの管理情報を取得します。

引　　数

ID (r)	word	スレッド ID
BUFF (w)	address	スレッドの管理情報を格納するバッファ

コ ー ル

```
        PEA       BUFF
        MOVE. W   ID, -(SP)
        DC. W     _GET_PR
        ADDQ. L   #6, SP
          :
BUFF:   DS. B     116
```



リターン　D0. L　　スレッド ID またはエラーコード

解　　説　指定された ID のスレッドの管理情報を BUFF にコピーします。
ID に−1 を指定し、BUFF+96 にスレッドの名前を指定すると、リターン値としてスレッドの ID が返り、管理情報が BUFF にコピーされます。
ID に−2 を指定すると、自分自身の ID がリターン値として返り、管理情報が BUFF にコピーされます。
以下に管理情報のフォーマットを示します。

```
DC. L   NEXT_PTR   *次のスレッドの管理情報格納領域
DC. B   WAIT_FLG   * nomal = $00／waid = $ff
DC. B   COUNT      *割り込みのたびに減算される
DC. B   MAX_CNT    * COUNT のプリセット値
DC. B   DOSCMD     * DOS コール番号
DC. L   PSP_ID     *プロセス ID
DC. L   USP
DC. L   D0, D1, D2, D3, D4, D5, D6, D7
DC. L   A0, A1, A2, A3, A4, A5, A6
```

```
DC.W    SR
DC.L    PC
DC.L    SSP
DC.W    INDOSF      *システム予約済
DC.L    INDOSP      *システム予約済
DC.L    BUFF        *タスク間通信バッファ
DS.B    16          *スレッドの名前
DC.L    WAIT_TIME   *待ち時間の残り（ミリ秒）
```

リターン値は正の数ならスレッドID、負の数ならエラーコードです。

suspend_pr

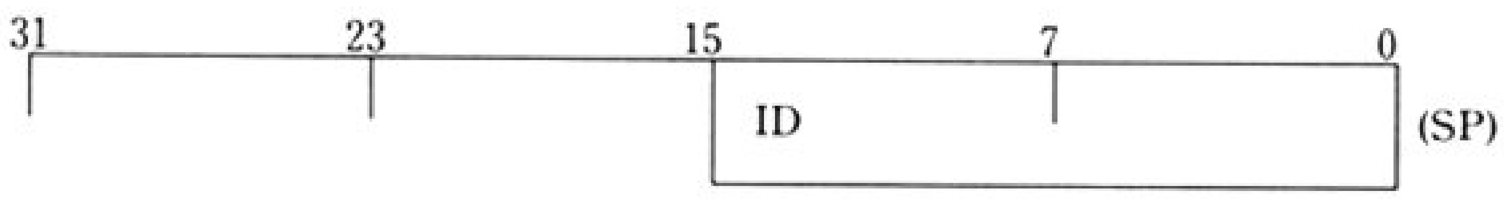
● suspend_pr (ID)　　DOS コール番号：$FFFB

機　　能　指定したスレッドを強制的にスリープ状態にします。

引　　数　ID (r)　　word　　スレッド ID

コ ー ル

```
MOVE. W   ID, -(SP)
DC. W     _SUSPEND_PR
ADDQ. L   #2, SP
```



リターン　D0. L

D0. L ＝ 0 ………………正常終了

D0. L ＜ 0 ………………エラーコード

D0. L ＝ −1 ……………スレッド自身のエラー

D0. L ＝ $FFFF00?? …… ID がおかしい（$??は ID の最大値）

解　　説　指定した ID のスレッドを、強制的にスリープ状態にします。

スリープ状態になったスレッドは、send_pr（$FFFD）で起こされるまでスリープします。

リターン値は、正常終了なら 0 を D0. L に返します。

負の数ならばエラーコードです。

ID がおかしい場合は $FFFF00?? を D0. L に返します。

$??が ID の最大値です。

スレッド自身のエラーなら−1 を D0. L に返します。

sleep_pr

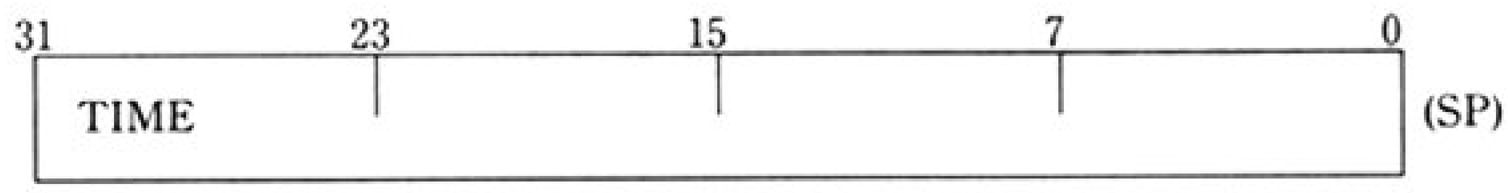
● sleep_pr (TIME)　　DOSコール番号：$FFFC

機　能　スリープ状態に入ります。

引　数　TIME (r)　long　待ち時間（ミリ秒単位）

コール

```
        MOVE. L    TIME, -(SP)
        DC. W      _SLEEP_PR
        ADDQ. L    #4, SP
```



リターン　D0. L　　スレッドの状態を示す値

D0. L ＝ −1………… 待ち時間が経過し、自分で起きた

D0. L ＝ −2………… 待ち時間が経過した後、send_pr（$FFFD）で起こされた

D0. L ＝ 上記以外……残り時間（send_pr（$FFFD）で起こされた）

解　説　スリープ状態に入ります。

TIMEには、待ち時間を指定します。

0を指定した場合は永久にスリープします。

スリープ状態に入ったスレッドは、send_pr（$FFFD）を用いて強制的に起こすことができます。

send_pr（$FFFD）のコマンドが$FFFBの場合、タスク間通信バッファは変化しません。それ以外のコマンドの場合は、バッファに次のように設定されます。

```
BUFF:   DC. L    LENGTH         *データ用バッファに書き込まれた
                                *バイト数
        DC. L    DATABUF_ADR    *データ用バッファの先頭アドレス
        DC. W    COMMAND        *コマンド番号
        DC. W    ID             *起こしてくれたスレッドのID
```

スリープする前に、データバッファの内容を処理してからデータバッファのアドレスとバイト数を設定し、IDに−1を設定してください。

これにより、他のスレッドからの通信を許可することになります。

スリープしていないときでも、send_pr（$FFF0）によりデータが送られる場合があります。その場合はスリープするとすぐに起こされ、設定した待ち時間をリターン値として返します。
リターン値（D0. L）は次の通りです。

- D0. L ＝ −1
  待ち時間が経過して自分で起きたことを表します。
  タスク間通信バッファの内容は変化しません。
- D0. L ＝ −2
  待ち時間が経過したが、suspend_pr（$FFFB）で止められていたためsend_pr（$FFFD）で起こされたことを表します。
  タスク間通信バッファは、send_pr（$FFFD）により変化します。
- D0. L ＝ 上記以外
  待ち時間は経過してないが、send_pr（$FFFD）で起こられたことを表します。
  この時戻り値は、ミリ秒単位で残り時間を表します。
  バッファの内容は、send_pr（$FFFD）により変化します。

send_pr

● send_pr (MY_ID, YOUR_ID, CMDNO, U_BUFF, U_LEN)

DOS コール番号：$FFFD

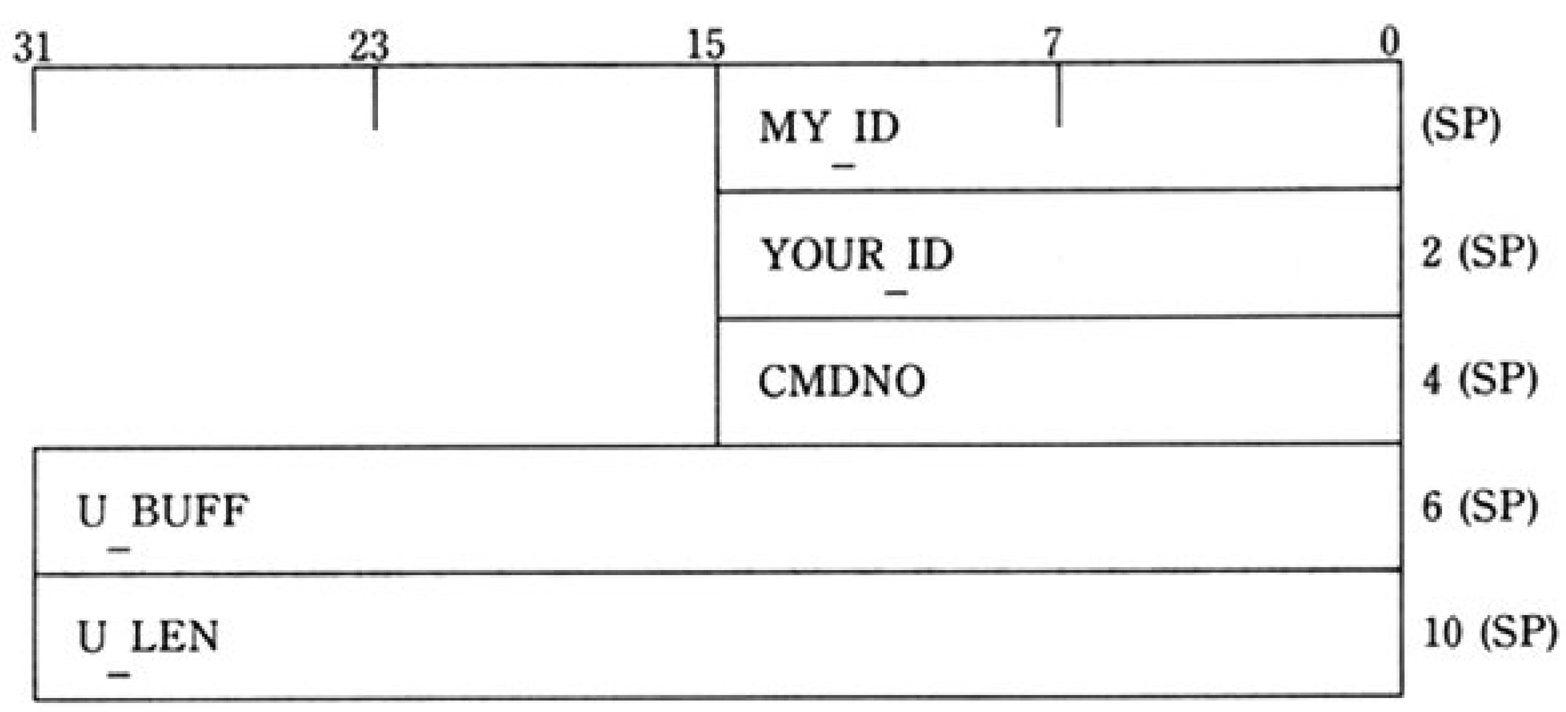
機　　能　指定したIDのスレッドにコマンドやデータを送り、スリープしていたら起こします。

引　　数

MY_ID (r)	word	自分のスレッドID
YOUR_ID (r)	word	相手のスレッドID
CMDNO (r)	word	送出するコマンド
U_BUFF (w)	address	送出するデータの格納領域
U_LEN (r)	long	送出するデータのバイト数

コ ー ル

```
MOVE. L     U_LEN, -(SP)
PEA         U_BUFF
MOVE. W     CMDNO, -(SP)
MOVE. W     YOUR_ID, -(SP)
MOVE. W     MY_ID, -(SP)
DC. W       _SEND_PR
LEA         14 (SP), SP
```



リターン　D0. L

D0. L ＝ 0 ……………正常終了
D0. L ＜ 0 ……………エラーコード
D0. L ＝ −28…………書き込みエラー
D0. L ＝ $FFFF00??…指定したIDが不正（$??はIDの最大値）
D0. L ＝ $800?????……指定したU_LENが不正（$?????はU_LENの最大値）

解　　説　指定したIDのスレッドにコマンドやデータを送り、スリープしていたら起こし

ます。
MY_ID は自分のスレッドの ID です。
YOUR_ID は、通信したい相手のスレッドの ID です。
CMDNO は通信の内容を表すワードの値で、システムで定められているコマンド以外はそれぞれのスレッド間で定義します。
システムで予約してあるコマンドは$FF??であり、以下に示す値が定義されています。

$FFF9：スレッドを消去するよう要求する
$FFFB：強制スリープ状態から起こすのみ
　　タスク間通信バッファは変化しない。
$FFFC：スリープするよう要求する
　　起こされたときこのコマンドが送られていたならば、すぐにYOUR_ID を－1にしてスリープすべきである。
　　通常、スリープしないでタスク間通信バッファを監視している場合に有効。
$FFFF：処理が終わったかどうか調べるためのコマンド
　　－28 が返れば、まだ処理中であることを示す。

コマンドやデータの転送には、タスク間通信バッファが使用されます。
タスク間通信バッファの構造を以下に示します。

```
BUFF:   DC.L    LENGTH        *データ用バッファのバイト数
        DC.L    DATABUF_ADR   *データ用バッファの先頭アドレス
        DC.W    COMMAND       *コマンド番号
        DC.W    ID            * ID (－1で通信許可)
```

引数の YOUR_ID で指定したスレッドの、タスク間通信バッファの YOUR_ID が（－1）に設定されていれば、そのバッファは書き込み可能なので、MY_ID とCMDNOをそれぞれ通信バッファのYOUR_ID、COMMANDに設定し、U_BUFF からのデータを U_LEN だけ、通信バッファの DATABUF_ADR で示されるデータバッファに書き込みます。
そして、U_LEN を通信バッファの LENGTH に設定します。
また、指定したスレッドがスリープしていたら起こします。

特に$FFFB のコマンドは特殊処理され、指定したスレッドを起こすだけで、タスク間通信バッファの YOUR_ID が（－1）でなくてもよく、その他のバッファも変化しません。
引数の U_LEN が、タスク間通信バッファの LENGTH より大きいときはエラ

ーとなります。

また、書き込み不可の場合もエラーとなります。

リターン値はエラーコードです。

正常終了の場合は0を返します。

エラーコードが$FFFF00??の場合は、指定したIDが不正であり、$??がIDの最大値を示します。

エラーコードが$800?????の場合は、指定したU_LENが不正であり、$?????がU_LENの最大値を示します。

書き込みエラーの場合は、－28を返します。

time_pr

● time_pr ( )　　DOS コール番号：$FFFE

機　　能	現在のタイマのカウンタの値を返します。
コール	DC. W　　_TIME_PR
リターン	D0. L　　現在のタイマのカウント値
解　　説	現在のタイマーのカウンタ値（ミリ秒単位）を D0. L に返します。 バックグラウンドで複数のスレッドが並行して動作している場合、プログラムで一定の時間を計るためには、どのスレッドが動作していても、一定して変化するカウンタが必要になります。 long での最大値を越えると 0 に戻るので、そのまま前回に返されたタイマのカウンタ値との引き算で、経過時間が分かります。

change_pr

● change_pr（ ）　　DOSコール番号：$FFFF

機　　能　バックグラウンドタスクの自分の実行権を放棄します。

コ ー ル　DC. W　　_CHANGE_PR

リターン　なし

解　　説　バックグラウンドタスクの自分の実行権を放棄します。
次のタスクに切り替わります。

★標準入出力を用いたサンプルプログラム

次のプログラムは、標準入力（デフォルトはキーボード）から入力した1文字を標準出力（デフォルトはディスプレイ画面）に4文字コピーします。

[CTRL]+[Z]または[CTRL]+[C]で終了します。

```
*-------------------------------------------
* 標準入力から入力した文字を 4個出力します
*
* '^Z' を入力すると終了します
*
* Function
*       $00: exit
*       $01: getchar
*       $02: putchar
*-------------------------------------------
        .include        doscall.mac

        .text
LOOP:
        move.w  #'?',-(sp)      * '?' を出力
        dc.w    _PUTCHAR
        addq.l  #2,sp

        dc.w    _GETCHAR        * 1 文字入力して
        move.w  d0,d1           * d1.w にデータを保存
        cmp.b   #$1a,d0         * $1a (^Z) なら EXIT へ
        beq     EXIT

        move.w  #':',-(sp)      * ':' を出力
        dc.w    _PUTCHAR
        addq.l  #2,sp

        move.w  #3,d2           * d2 にループ回数をセット
TIME3:
        move.w  d1,-(sp)        * d1 に保存したデータを出力
        dc.w    _PUTCHAR
        addq.l  #2,sp
        dbra    d2,TIME3        * d2 から 1 引いて >0 なら TIME3 へ

        move.w  #$0d,-(sp)      * $0d (CR) を出力
        dc.w    _PUTCHAR
        addq.l  #2,sp

        move.w  #$0a,-(sp)      * $0a (LF) を出力
        dc.w    _PUTCHAR
        addq.l  #2,sp

        bra     LOOP            * LOOP へジャンプ

EXIT:
        dc.w    _EXIT           * プロセスの終了
```

★ファイルのオープン、クローズを用いたサンプルプログラム

次のプログラムは、コピー元とコピー先のファイル名をたずねてきます。
その後、指定したファイルを指定した名前でコピーします。
存在しないファイル名や許されないファイル名を指定するとエラーを表示します。
ファイルを1つコピーするか、エラーが発生すると終了します。

```
*---------------------------------------------
*  ディスクファイルのコピーを行います
*  プロンプトに従ってファイル名を入力してください
*
*  Function
*       $3c: create
*       $3d: open
*       $3e: close
*       $3f: read
*       $40: write
*---------------------------------------------
        .include        doscall.mac

        .text

MAIN:
        pea     SFILEP          * プロンプト(sorc用)のアドレスをプッシュ
        pea     SFILE           * コピー元ファイル名のアドレスをプッシュ
        bsr     GETFNAM         * コピー元ファイル名の入力
        addq.l  #8,sp           * スタックレベルの調整

        pea     DFILEP          * プロンプト(dist用)のアドレスをプッシュ
        pea     DFILE           * コピー先ファイル名のアドレスをプッシュ
        bsr     GETFNAM         * コピー先ファイル名の入力
        addq.l  #8,sp           * スタックレベルの調整

        bsr     SFOPEN          * コピー元ファイルオープン
        bsr     DFOPEN          * コピー先ファイルオープン
NEXT:
        bsr     READREC         * 1セクタ読み込み
        cmp.l   #1024,d0        * 読み込んだバイト数が 1024 以下なら
        blt     CPYEND          * CPYEND へジャンプ
        bsr     WRITEREC        * 1セクタ書き出し
        bra     NEXT            * NEXT へジャンプ
CPYEND:
        bsr     WRITEREC        * 1セクタ書き出し
        bsr     SFCLOSE         * コピー元ファイルクローズ
        bsr     DFCLOSE         * コピー先ファイルクローズ
        dc.w    _EXIT           * プロセスの終了

*------ End of Main Prg. -------------
```

```
*-----------------------------------
* Get File Name
*       0(sp) Return Address
*       8(sp) Prompt Address
*       4(sp) File Name Address
*-----------------------------------
GETFNAM:
        move.l  8(sp),-(sp)     * プロンプト出力
        dc.w    _PRINT
        addq.l  #4,sp

        move.l  4(sp),-(sp)     * ファイルネームの入力
        dc.w    _GETS
        addq.l  #4,sp

        move.w  #$0a,-(sp)      * $0a (LF) 出力
        dc.w    _PUTCHAR
        addq.l  #2,sp

        rts

        SFOPEN:
                move.w  #0,-(sp)        * 読み出し専用モード
                lea     SFILE,a0        * コピー元ファイル名のアドレスをプッシュ
                adda.l  #2,a0           *
                move.l  a0,-(sp)        *
                dc.w    _OPEN           * コピー元ファイルオープン
                addq.l  #6,sp           * スタックレベル調整
                blt     SFOPN_ERR       * エラーならば SFOPN_ERR へジャンプ
                move.w  d0,SF_HNDL      * ファイルハンドルを SF_HNDL にセーブ
                rts

        DFOPEN:
                move.w  #$20,-(sp)      * アトリビュート
                lea     DFILE,a0        * コピー先ファイル名のアドレスをプッシュ
                adda.l  #2,a0           *
                move.l  a0,-(sp)        *
                dc.w    _CREATE         * コピー先ファイルオープン(クリエイト)
                addq.l  #6,sp           * スタックレベル調整
                blt     DFOPN_ERR       * エラーならば DFOPN_ERR へジャンプ
                move.w  d0,DF_HNDL      * ファイルハンドルを DF_HNDL にセーブ
                rts


        SFCLOSE:
                move.w  SF_HNDL,-(sp)   * コピー元ファイルハンドルをプッシュ
                dc.w    _CLOSE          * コピー元ファイルクローズ
                addq.l  #2,sp           * スタックレベル調整
                blt     SFC_ERR         * エラーならば SFC_ERR へジャンプ
                rts

        DFCLOSE:
                move.w  DF_HNDL,-(sp)   * コピー先ファイルハンドルをプッシュ
                dc.w    _CLOSE          * コピー先ファイルをクローズ
                addq.l  #2,sp           * スタックレベル調整
                blt     DFC_ERR         * エラーならば DFC_ERR へジャンプ
                rts


        READREC:                        ****  1セクタリード ****
                move.l  #1024,-(sp)     * 読み込むバイト数をプッシュ
                pea     TMPBUF          * 読み込むバッファのアドレスをプッシュ
                move.w  SF_HNDL,-(sp)   * コピー元のファイルハンドルをプッシュ
                dc.w    _READ           * 1セクタリード
                lea     10(sp),sp       * スタックレベル調整
                blt     READ_ERR        * エラーならば READ_ERR へジャンプ
                rts
```

```
WRITEREC:                                   **** 1セクタライト ****
        move.l  d0,-(sp)                    * d0.l の書き込むバイト数をプッシュ
        pea     TMPBUF                      * データバッファアドレスをプッシュ
        move.w  DF_HNDL,-(sp)               * コピー先のファイルハンドルをプッシュ
        dc.w    _WRITE                      * ディスクライト
        lea     10(sp),sp                   * スタックレベル調整
        blt     WRITE_ERR                   * エラーならば WRITE_ERR
        rts


SFOPN_ERR:                                  **** sorc. open error ****
        lea     E_SFO,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT


DFOPN_ERR:                                  **** dist. create error ****
        lea     E_DFO,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT

SFC_ERR:                                    **** sorc. close error ****
        lea     E_SFC,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT

DFC_ERR:                                    **** dist. close error ****
        lea     E_DFC,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT

READ_ERR:                                   **** read error ****
        lea     E_READ,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT

WRITE_ERR:                                  **** write error ****
        lea     E_WRITE,a0
        move.l  a0,-(sp)
        dc.w    _PRINT
        addq.l  #4,sp
        dc.w    _EXIT
```

```
**********************************
*    メッセージ データ
**********************************
SFILEP  dc.b    " コピー元 ファイル名を入力してください > ",0
DFILEP  dc.b    "新しく作るファイル名を入力してください > ",0

E_SFO   dc.b    "Can't Open Source File ",0
E_DFO   dc.b    "Can't Open Dist.  File ",0
E_SFC   dc.b    "Can't Close ",0
E_DFC   dc.b    "Can't Close ",0
E_READ  dc.b    "Read Error ",0
E_WRITE dc.b    "Write Error ",0

*********************************
* ファイル 情報
*********************************
SFILE   dc.b    80,0
        ds.b    81
DFILE   dc.b    80,0
        ds.b    81
        .even
SF_HNDL dc.w    0
DF_HNDL dc.w    0

********************************
* データバッファ
********************************
TMPBUF:
        ds.b    1024
```

第3章

IOCS コール

DOS コールと並び、X68000で使用できるファンクションコールの一つに、このIOCSコールがあります。
本書第2章で解説しているDOSコールがHuman68k内部のサブルーチンをコールしているのに対し、IOCSコールはX68000内のROMに書かれたサブルーチンをコールしています。

3.1 IOCSコール一覧

IOCS コールの一覧表を示します。
はじめにコード順、次が名前順です。
また、IOCS コールで使用する以外の TRAP ベクタについても示します。

IOCS コール(コード順)

コード	名　前	機　能	ページ
$00	B_KEYINP	1 キーデータ入力	252
$01	B_KEYSNS	キーバッファチェック	254
$02	B_SFTSNS	シフトキー状態チェック	256
$03	KEY_INIT	キー関係の初期化	258
$04	BITSNS	キー状態チェック	259
$0C	TVCTRL	テレビコントロール	261
$0D	LEDMOD	LED モードの指定	263
$0E	TGUSEMD	画面の使用状態を OS に知らせる	265
$0F	DEFCHR	外字定義	267
$10	CRTMOD	CRT モードの指定	269
$11	CONTRAST	コントラストの設定	271
$12	HSVTORGB	HSV → RGB 変換	272
$13	TPALET	テキストパレットの指定	274
$15	TCOLOR	テキストプレーン指定	276
$16	FNTADR	漢字フォントアドレスを求める	278
$17	VRAMGET	VRAM からの読み込み	282
$18	VRAMPUT	VRAM への書き込み	284
$19	FNTGET	フォントデータ読み込み	285
$1A	TEXTGET	パターンゲット	290
$1B	TEXTPUT	パターンプット	292
$1C	CLIPPUT	パターンプット（クリッピング処理をする）	294
$1D	SCROLL	表示範囲の設定／チェック	296
$1E	B_CURON	カーソル表示	298
$1F	B_CUROFF	カーソル非表示	299
$20	B_PUTC	1 文字出力（漢字可）	300
$21	B_PRINT	文字列出力	301
$22	B_COLOR	文字属性設定	302

IOCS コール(コード順)

コード	名　前	機　能	ページ
$23	B_LOCATE	カーソル移動（座標指定）	304
$24	B_DOWN_S	カーソル移動１行下（最下行でスクロール）	306
$25	B_UP_S	カーソル移動１行上（先頭行でスクロール）	307
$26	B_UP	カーソル移動n行上	308
$27	B_DOWN	カーソル移動n行下	309
$28	B_RIGHT	カーソル移動n桁右	310
$29	B_LEFT	カーソル移動n桁左	311
$2A	B_CLR_ST	画面の複数行にわたって消去	312
$2B	B_ERA_ST	画面の現在行の複数桁消去	313
$2C	B_INS	インサート（行単位）	314
$2D	B_DEL	デリート（行単位）	315
$2E	B_CONSOL	表示範囲の指定	316
$2F	B_PUTMES	座標指定文字列出力	318
$30	SET232C	RS-232Cパラメータ設定	320
$31	LOF232C	RS-232C受信バッファのデータ数を求める	322
$32	INP232C	RS-232C受信データを得る	323
$33	ISNS232C	RS-232C受信データをチェック	324
$34	OSNS232C	RS-232Cが送信可能かどうかをチェック	325
$35	OUT232C	RS-232Cに送信を行う	326
$3B	JOYGET	ジョイスティックポートのデータを読み込む	327
$3C	INIT_PRN	プリンタポートの初期化	329
$3D	SNSPRN	プリンタ出力可能かどうかをチェック	330
$3E	OUTLPT	プリンタ直接出力	331
$3F	OUTPRN	プリンタ出力（漢字も可能）	332
$40	B_SEEK	ディスクのシーク（2HD）	334
$40	B_SEEK	ディスクのシーク（ハードディスク）	337
$41	B_VERIFY	ディスクデータのベリファイ（2HD）	339
$41	B_VERIFY	ディスクデータのベリファイ(ハードディスク)	341
$42	B_READDI	診断のためのディスクからの読み取り	343
$43	B_DSKINI	ディスクの初期化（2HD）	345
$43	B_DSKINI	ディスクの初期化（ハードディスク）	347
$44	B_DRVSNS	ディスクの状態のチェック	348
$45	B_WRITE	ディスクへの書き込み	349
$46	B_READ	ディスクからの読み込み	351
$47	B_RECALI	ディスクのリキャリブレイト（2HD）	353
$47	B_RECALI	ディスクのリキャリブレイト(ハードディスク)	355

3.1　IOCSコール一覧

IOCS コール(コード順)

コード	名　前	機　能	ページ
$49	B_WRITED	破損データ書き込み（2HD）	356
$4B	B_BADFMT	破損トラックを使用不能にする（ハードディスク）	357
$4C	B_READDL	破損データの読み込み（2HD）	358
$4D	B_FORMAT	ディスクのフォーマット（2HD）	360
$4D	B_FORMAT	ディスクのフォーマット（ハードディスク）	362
$4E	B_DRVCHK	ドライブ状態設定（2HD）	363
$4F	B_EJECT	イジェクト（2HD）	365
$4F	B_EJECT	シッピング（ハードディスク）	366
$50	DATEBCD	日付データの変換　バイナリ→BCD	367
$51	DATESET	日付の設定	369
$52	TIMEBCD	時刻データの変換　バイナリ→BCD	371
$53	TIMESET	時刻の設定	373
$54	DATEGET	日付の読み込み（BCD）	374
$55	DATEBIN	日付データの変換　BCD→バイナリ	375
$56	TIMEGET	時刻の読み込み（BCD）	377
$57	TIMEBIN	時刻データの変換　BCD→バイナリ	378
$58	DATECNV	日付データの変換　文字列→バイナリ	380
$59	TIMECNV	時刻データの変換　文字列→バイナリ	382
$5A	DATEASC	日付データの変換　バイナリ→文字列	384
$5B	TIMEASC	時刻データの変換　バイナリ→文字列	386
$5C	DAYASC	曜日データの変換　バイナリ→文字列	388
$5D	ALARMMOD	アラームの禁止／許可設定・チェック	389
$5E	ALARMSET	アラームの時間と処理アドレス設定	390
$5F	ALARMGET	アラームの時間と処理アドレスチェック	392
$60	ADPCMOUT	ADPCMのデータ出力（ブロック）	393
$61	ADPCMINP	ADPCMのデータ入力（ブロック）	395
$62	ADPCMAOT	ADPCMのデータ出力（アレイチェーン）	397
$63	ADPCMAIN	ADPCMのデータ入力（アレイチェーン）	399
$64	ADPCMLOT	ADPCMのデータ出力（リンクチェーン）	401
$65	ADPCMLIN	ADPCMのデータ入力（リンクチェーン）	403
$66	ADPCMSNS	ADPCMの状態チェック	405
$67	ADPCMMOD	ADPCMの実行制御	407
$68	OPMSET	OPM（FM音源）のレジスタ設定	408
$69	OPMSNS	OPM（FM音源）の状態チェック	409
$6A	OPMINTST	OPM（FM音源）の割り込みの設定	410

IOCS コール(コード順)

コード	名　前	機　能	ページ
$6B	TIMERDST	MFP の TIMER-D による割り込み制御	411
$6C	VDISPST	垂直同期による割り込み制御	413
$6D	CRTCRAS	ラスター走査による割り込み制御	415
$6E	HSYNCST	水平同期信号立下がりによる割り込み制御	416
$6F	PRNINTST	プリンタによる割り込み制御	417
$70	MS_INIT	マウスの初期化	418
$71	MS_CURON	マウスカーソルの表示	419
$72	MS_CUROF	マウスカーソルの非表示	420
$73	MS_STAT	マウスカーソルの表示モードチェック	421
$74	MS_GETDT	マウスの移動量／ボタンの状態チェック	422
$75	MS_CURGT	マウスの現在座標チェック	424
$76	MS_CURST	マウスの座標指定	425
$77	MS_LIMIT	マウスの移動範囲指定	426
$78	MS_OFFTM	マウスボタンを離すまでの時間のチェック	428
$79	MS_ONTM	マウスボタンを押すまでの時間のチェック	429
$7A	MS_PATST	マウスカーソルパターンを定義	430
$7B	MS_SEL	マウスカーソルパターンを指定	432
$7C	MS_SEL2	マウスカーソルアニメーション	433
$7D	SKEY_MOD	ソフトキーボードの制御	435
$7E	DENSNS	電卓の制御	437
$7F	ONTIME	電源起動後の経過時間を返す	438
$80	B_INTVCS	ベクタの設定	439
$81	B_SUPER	スーパーバイザ／ユーザーモードの切り換え	440
$82	B_BPEEK	指定アドレスから1バイト読み込み	442
$83	B_WPEEK	指定アドレスから1ワード読み込み	443
$84	B_LPEEK	指定アドレスから1ロングワード読み込み	444
$85	B_MEMSTR	指定アドレスから複数バイト読み込み	445
$86	B_BPOKE	指定アドレスへ1バイト書き込み	446
$87	B_WPOKE	指定アドレスへ1ワード書き込み	447
$88	B_LPOKE	指定アドレスへ1ロングワード書き込み	448
$89	B_MEMSET	指定アドレスへら複数バイト書き込み	449
$8A	DMAMOVE	DMA 転送（ブロック）	450
$8B	DMAMOV_A	DMA 転送（アレイチェーン）	452
$8C	DMAMOV_L	DMA 転送（リンクチェーン）	454
$8D	DMAMODE	DMA 状態チェック	456
$8E	BOOTINF	ブート情報を返す	457

IOCS コール(コード順)

コード	名　　前	機　　能	ページ
$8F	ROMVER	ROM バージョンを返す	459
$90	G_CLR_ON	グラフィック画面の初期化と表示モードの設定	460
$94	GPALET	グラフィックのパレット設定	461
$A0	SFTJIS	シフト JIS コード→ JIS 漢字コード変換	462
$A1	JISSFT	JIS 漢字コード→シフト JIS コード変換	464
$A2	AKCONV	ANK →シフト JIS コード変換	466
$A3	RMACNV	ローマ字→ ANK カナ変換	468
$A4	DAKJOB	濁点処理（全角文字列）	470
$A5	HANJOB	半濁点（全角文字列）	471
$AE	OS_CURON	カーソル表示（IOCS$20～$2F に有効）	472
$AF	OS_CUROF	カーソル表示（IOCS$20～$2F に有効）	473
$B1	APAGE	グラフィック画面の書き込みページ指定	474
$B2	VPAGE	グラフィック画面の表示ページ指定	476
$B3	HOME	グラフィック画面の表示位置設定	477
$B4	WINDOW	グラフィック画面のウィンドウ設定	479
$B5	WIPE	グラフィック画面の画面クリア	482
$B6	PSET	グラフィック画面のポイントセット	483
$B7	POINT	グラフィック画面のポイントゲット	484
$B8	LINE	グラフィック画面のライン	486
$B9	BOX	グラフィック画面のボックス	488
$BA	FILL	グラフィック画面のボックスフィル	490
$BB	CIRCLE	グラフィック画面のサークル	492
$BC	PAINT	グラフィック画面のペイント	494
$BD	SYMBOL	グラフィック画面のシンボル（クリッピングなし）	496
$BE	GETGRM	グラフィック画面の画面ゲット	498
$BF	PUTGRM	グラフィック画面の画面プット	500
$C0	SP_INIT	スプライト画面の初期化	502
$C1	SP_ON	スプライト画面の表示	503
$C2	SP_OFF	スプライト画面の非表示	504
$C3	SP_CGCLR	スプライト／バックグラウンドのクリア	505
$C4	SP_DEFCG	スプライト／バックグラウンドの定義	506
$C5	SP_GTPCG	スプライト／バックグラウンド読み込み	508
$C6	SP_REGST	スプライトレジスタの設定	510
$C7	SP_REGGT	スプライトレジスタのチェック	512

IOCS コール(コード順)

コード	名　前	機　能	ページ
$C8	BGSCRLST	バックグラウンドスクロールレジスタ設定	514
$C9	BGSCRLGT	〃　チェック	516
$CA	BGCTRLST	バックグラウンドコントロールレジスタ設定	517
$CB	BGCTRLGT	〃　チェック	519
$CC	BGTEXTCL	バックグラウンドテキストクリア	520
$CD	BGTEXTST	バックグラウンドテキスト設定	522
$CE	BGTEXTGT	バックグラウンドテキストチェック	524
$CF	SPALET	スプライトパレット設定・チェック	526
$FD	ABORTRST	アボート処理（OS）	528
$FF	ABORTJOB	アボート処理（OS）	529

IOCS コール(名前順)

コード	名　前	機　能	ページ
$63	ADPCMAIN	ADPCM のデータ入力（アレイチェーン）	399
$62	ADPCMAOT	ADPCM のデータ出力（アレイチェーン）	397
$61	ADPCMINP	ADPCM のデータ入力（ブロック）	395
$65	ADPCMLIN	ADPCM のデータ入力（リンクチェーン）	403
$64	ADPCMLOT	ADPCM のデータ出力（リンクチェーン）	401
$67	ADPCMMOD	ADPCM の実行制御	407
$60	ADPCMOUT	ADPCM のデータ出力（ブロック）	393
$66	ADPCMSNS	ADPCM の状態チェック	405
$A2	AKCONV	ANK →シフト JIS コード変換	466
$5D	ALARMMOD	アラームの禁止／許可設定・チェック	389
$5E	ALARMSET	アラームの時間と処理アドレス設定	390
$5F	ALARMGET	アラームの時間と処理アドレスチェック	392
$B1	APAGE	グラフィック画面の書き込みページ指定	474
$CB	BGCTRLGT	バックグラウンドコントロールレジスタチェック	519
$CA	BGCTRLST	バックグラウンドコントロールレジスタ設定	517
$C9	BGSCRLGT	バックグラウンドスクロールレジスタチェック	516
$C8	BGSCRLST	バックグラウンドスクロールレジスタ設定	514
$CC	BGTEXTCL	バックグラウンドテキストクリア	520
$CE	BGTEXTGT	バックグラウンドテキストチェック	524
$CD	BGTEXTST	バックグラウンドテキスト設定	522

IOCS コール(名前順)

コード	名　前	機　能	ページ
$04	BITSNS	キー状態チェック	259
$8E	BOOTINF	ブート情報を返す	457
$B9	BOX	グラフィック画面のボックス	488
$4B	B_BADFMT	破損トラックを使用不能にする（ハードディスク）	357
$82	B_BPEEK	指定アドレスから 1 バイト読み込み	442
$86	B_BPOKE	指定アドレスへ 1 バイト書き込み	446
$2A	B_CLR_ST	画面の複数行にわたって消去	312
$22	B_COLOR	文字属性設定	302
$2E	B_CONSOL	表示範囲の指定	316
$1F	B_CUROFF	カーソル非表示	299
$1E	B_CURON	カーソル表示	298
$2D	B_DEL	デリート（行単位）	315
$27	B_DOWN	カーソル移動 n 行下	309
$24	B_DOWN_S	カーソル移動 1 行下（最下行でスクロール）	306
$4E	B_DRVCHK	ドライブ状態設定（2HD）	363
$44	B_DRVSNS	ディスクの状態のチェック	348
$43	B_DSKINI	ディスクの初期化（2HD）	345
$43	B_DSKINI	ディスクの初期化（ハードディスク）	347
$4F	B_EJECT	イジェクト（2HD）	365
$4F	B_EJECT	シッピング（ハードディスク）	366
$2B	B_ERA_ST	画面の現在行の複数桁消去	313
$4D	B_FORMAT	ディスクのフォーマット（2HD）	360
$4D	B_FORMAT	ディスクのフォーマット（ハードディスク）	362
$2C	B_INS	インサート（行単位）	314
$80	B_INTVCS	ベクタの設定	439
$00	B_KEYINP	1 キーデータの入力	252
$01	B_KEYSNS	キーバッファチェック	254
$29	B_LEFT	カーソル移動 n 桁左	311
$23	B_LOCATE	カーソル移動（座標指定）	304
$84	B_LPEEK	指定アドレスから 1 ロングワード読み込み	444
$88	B_LPOKE	指定アドレスへ 1 ロングワード書き込み	448
$89	B_MEMSET	指定アドレスへ複数バイト書き込み	449
$85	B_MEMSTR	指定アドレスから複数バイト読み込み	445
$21	B_PRINT	文字列出力	301
$20	B_PUTC	1 文字出力（漢字可）	300

IOCS コール(名前順)

コード	名　前	機　能	ページ
$2F	B_PUTMES	座標指定文字列出力	318
$46	B_READ	ディスクからの読み込み	351
$42	B_READDI	診断のためのディスクからの読み取り	343
$4C	B_READDL	破損データの読み込み (2HD)	358
$47	B_RECALI	ディスクのリキャリブレイト (2HD)	353
$47	B_RECALI	ディスクのリキャリブレイト (ハードディスク)	355
$28	B_RIGHT	カーソル移動n桁右	310
$40	B_SEEK	ディスクのシーク (2HD)	334
$4D	B_SEEK	ディスクのシーク (ハードディスク)	337
$02	B_SFTSNS	シフトキー状態チェック	256
$81	B_SUPER	スーパーバイザ/ユーザーモードの切り換え	440
$26	B_UP	カーソル移動n行上	308
$25	B_UP_S	カーソル移動1行上 (先頭行でスクロール)	307
$41	B_VERIFY	ディスクデータのベリファイ (2HD)	339
$41	B_VERIFY	ディスクデータのベリファイ (ハードディスク)	341
$83	B_WPEEK	指定アドレスから1ワード読み込み	443
$87	B_WPOKE	指定アドレスへ1ワード書き込み	447
$45	B_WRITE	ディスクへの書き込み	349
$49	B_WRITED	破損データ書き込み (2HD)	356
$BB	CIRCLE	グラフィック画面のサークル	492
$1C	CLIPPUT	パターンプット (クリッピング処理をする)	294
$11	CONTRAST	コントラストの設定	271
$6D	CRTCRAS	ラスター走査による割り込み制御	415
$10	CRTMOD	CRT モードの指定	269
$A4	DAKJOB	濁点処理 (全角文字列)	470
$5A	DATEASC	日付データの変換　バイナリ→文字列	384
$50	DATEBCD	日付データの変換　バイナリ→ BCD	367
$55	DATEBIN	日付データの変換　BCD →バイナリ	375
$58	DATECNV	日付データの変換　文字列→バイナリ	380
$54	DATEGET	日付の読み込み (BCD)	374
$51	DATESET	日付の設定	369
$5C	DAYASC	曜日データの変換　バイナリ→文字列	388

IOCSコール(名前順)

コード	名　前	機　能	ページ
$0F	DEFCHR	外字定義	267
$7E	DENSNS	電卓の制御	437
$8D	DMAMODE	DMA 状態チェック	456
$8B	DMAMOV_A	DMA 転送（アレイチェーン）	452
$8A	DMAMOVE	DMA 転送（ブロック）	450
$8C	DMAMOV_L	DMA 転送（リンクチェーン）	454
$BA	FILL	グラフィック画面のボックスフィル	490
$16	FNTADR	漢字フォントアドレスを求める	278
$19	FNTGET	フォントデータ読み込み	285
$BE	GETGRM	グラフィック画面の画面ゲット	498
$94	GPALET	グラフィックのパレット設定	461
$90	G_CLR_ON	グラフィック画面の初期化と表示モードの設定	460
$A5	HANJOB	半濁点（全角文字列）	471
$B3	HOME	グラフィック画面の表示位置設定	477
$12	HSVTORGB	HSV → RGB 変換	272
$6E	HSYNCST	水平同期信号立下がりによる割り込み制御	416
$3C	INIT_PRN	プリンタポートの初期化	329
$32	INP232C	RS-232C 受信データを得る	323
$33	ISNS232C	RS-232C 受信データをチェック	324
$A1	JISSFT	JIS 漢字コード→シフト JIS コード変換	464
$3B	JOYGET	ジョイスティックポートのデータを読み込む	327
$03	KEY_INIT	キー関係の初期化	258
$0D	LEDMOD	LED モードの指定	263
$B8	LINE	グラフィック画面のライン	486
$31	LOF232C	RS-232C 受信バッファのデータ数を求める	322
$75	MS_CURGT	マウスの現在座標チェック	424
$72	MS_CUROF	マウスカーソルの非表示	420
$71	MS_CURON	マウスカーソルの表示	419
$76	MS_CURST	マウスの座標指定	425
$74	MS_GETDT	マウスの移動量／ボタンの状態チェック	422
$70	MS_INIT	マウスの初期化	418
$77	MS_LIMIT	マウスの移動範囲指定	426
$78	MS_OFFTM	マウスボタンを離すまでの時間のチェック	428
$79	MS_ONTM	マウスボタンを押すまでの時間のチェック	429
$7A	MS_PATST	マウスカーソルパターンを定義	430

IOCS コール(名前順)

コード	名　前	機　能	ページ
$7B	MS_SEL	マウスカーソルパターンを指定	432
$7C	MS_SEL2	マウスカーソルアニメーション	433
$73	MS_STAT	マウスカーソルの表示モードチェック	421
$7F	ONTIME	電源起動後の経過時間を返す	438
$6A	OPMINTST	OPM（FM音源）の割り込みの設定	410
$68	OPMSET	OPM（FM音源）のレジスタ設定	408
$69	OPMSNS	OPM（FM音源）の状態チェック	409
$34	OSNS232C	RS-232Cが受信可能かどうかをチェック	325
$AF	OS_CUROF	カーソル非表示（IOCS$20～$2Fに有効）	473
$AE	OS_CURON	カーソル表示（IOCS$20～$2Fに有効）	472
$35	OUT232C	RS-232Cに送信を行う	326
$3E	OUTLPT	プリンタ直接出力	331
$3F	OUTPRN	プリンタ出力（漢字も可能）	332
$BC	PAINT	グラフィック画面のペイント	494
$B7	POINT	グラフィック画面のポイントゲット	484
$6F	PRNINTST	プリンタによる割り込み制御	417
$B6	PSET	グラフィック画面のポイントセット	483
$BF	PUTGRM	グラフィック画面の画面プット	500
$FD	RESERVED	アボート処理（OS）	528
$FF	RESERVED	アボート処理（OS）	529
$A3	RMACNV	ローマ字→ANKカナ変換	468
$8F	ROMVER	ROMバージョンを返す	459
$1D	SCROLL	表示範囲の設定／チェック	296
$30	SET232C	RS-232Cパラメータ設定	320
$A0	SFTJIS	シフトJIS→JIS変換	462
$7D	SKEY_MOD	ソフトキーボードの制御	435
$3D	SNSPRN	プリンタ出力可能かどうかをチェック	330
$CF	SPALET	スプライトパレット設定・チェック	526
$C3	SP_CGCLR	スプライト／バックグラウンドのクリア	505
$C4	SP_DEFCG	スプライト／バックグラウンドの定義	506
$C5	SP_GTPCG	スプライト／バックグラウンドの読み込み	508
$C0	SP_INIT	スプライト画面の初期化	502
$C2	SP_OFF	スプライト画面の非表示	504
$C1	SP_ON	スプライト画面の表示	503

IOCS コール（名前順）

コード	名　前	機　能	ページ
$C7	SP_REGGT	スプライトレジスタのチェック	512
$C6	SP_REGST	スプライトレジスタの設定	510
$BD	SYMBOL	グラフィック画面のシンボル（クリッピングなし）	496
$15	TCOLOR	テキストプレーン指定	276
$1A	TEXTGET	パターンゲット	290
$1B	TEXTPUT	パターンプット	292
$0E	TGUSEMD	画面の使用状態を OS に知らせる	265
$5B	TIMEASC	時刻データの変換　バイナリ→文字列	386
$52	TIMEBCD	時間データの変換　バイナリ→ BCD	371
$57	TIMEBIN	時刻データの変換　BCD →バイナリ	378
$59	TIMECNV	時刻データの変換　文字列→バイナリ	382
$56	TIMEGET	時刻の読み込み（BCD）	377
$6B	TIMERDST	MFP の TIMER-D による割り込み制御	411
$53	TIMESET	時刻の設定	373
$13	TPALET	テキストパレットの指定	274
$0C	TVCTRL	テレビコントロール	261
$6C	VDISPST	垂直同期による割り込み制御	413
$B2	VPAGE	グラフィック画面の表示ページ指定	476
$17	VRAMGET	VRAM からの読み込み	282
$18	VRAMPUT	VRAM への書き込み	284
$B4	WINDOW	グラフィック画面のウィンドウ設定	479
$B5	WIPE	グラフィック画面の画面クリア	482

●参考資料：TRAP15以外のシステム TRAP

■ TRAP8……システム予約

■ TRAP9……OS デバッガ用

■ TRAP10……リセット／フロント SW／外部 SW／ソフト OFF 処理

入力 D0. L ＝ $58363801……フロント SW

＝ $58363802……外部 SW

＝ $58363803……ソフト OFF

＝ 上記以外………リセット

■ TRAP11……[BREAK]キー処理

入力 D0. B のビット 0 ＝ 0……[BREAK]（[CTRL]+[C]）

＝ 1……[SHIFT]+[BREAK]（[CTRL]+[S]）

■ TRAP12……[COPY]キー処理

入力 D0. B ＝ 0……縮小コピー（1 色）

＝ 1……拡大コピー（1 色）

＝ 2 または 3……拡大コピー（4 色）

■ TRAP13……[CTRL]+[C]処理

■ TRAP14……エラー表示と処理

入力 D7. W ＝ エラーコード

3.2 IOCSコールの詳細

3.2.1 IOCSコールの使い方

IOCSコールは、MPU68000に特有の命令であるTRAP命令を使って呼び出します。
TRAP命令は、MPU68000の例外処理を起動する命令の一つで、特にシステムで予約されたソフトウェア例外処理を行う場合に使用します。
X68000のIOCSコールには、数あるシステムTRAPのうちからソフトウェア例外処理の15番を使用します（TRAP15)。
レジスタD0にIOCSコール番号をセットし、その後TRAP15を実行すると、システムは例外処理ベクタ番号$2Fを発生させ、これによって$0BCに書かれているアドレスに分岐し、ここでD0のIOCSコール番号を確かめてから、各IOCSコールのサブルーチンに分岐するのです。
実際の実行例を次に示します。

```
MOVEQ. L    # no, D0    * noはIOCSコール番号
TRAP        # 15
```

あるいはマクロを使用して、

```
include     iocscall. mac
              :
IOCS        _B_KEYINP
```

リターン値のあるものは、リターン時にレジスタや指定したメモリなどに値を返します。
IOCSで使用するメモリは、以下のようになっています。

$000000～$0003FF ＝ 割り込み／TRAP／そのほか、システム用
$000400～$0007FF ＝ IOCSコール処理テーブル用
$000800～$000FFF ＝ IOCSワーク
$001000～$0017FF ＝ システム用ワーク
$001800～$001BFF ＝ DOSコール処理テーブル用
$001C00～$001FFF ＝ Human68kワーク

\$002000～\$0023FF	=	2HD用IPL、またはHD用IPL MENUプログラム
\$002400～\$0027FF	=	HD用IPLプログラム
\$00????～\$0067FF	=	Human68kスタック（SSP）
\$006800～???????	=	Human68k

3.2.2 IOCSコールの解説方法

3.2.3以降では、次のような形式でIOCSコールを解説します。

●名　前	**IOCSコール番号：\$??**
機　　能	機能の解説
入　　力	入力パラメータの解説
リターン	リターン値
解　　説	機能の詳細な解説
プログラム例	IOCSコールを使った小さなプログラム、サブルーチンなどの例

・名前

説明するIOCSコールの名前です。
名前は機能を示すようになっています。

・IOCSコール番号

IOCSコールの機能を示す番号です。
16進数2桁で書かれています。
入力パラメータのD0.Lに代入します。

・機能

機能の概要を示します。

・入力

コールで必要な入力パラメータを示します。
D0.LにはIOCSコール番号を入力します。
A0レジスタは入力に指定することができません。
A0以外のレジスタについてはそれぞれのIOCSコールによって異なります。

・リターン

IOCS コールの結果を、いくつかのレジスタに返す場合があります。
それぞれのIOCS コールにより異なります。
原則としてここで指定した以外のレジスタ（D0を除く）は壊れません。
レジスタ図のうち□となっているのは、入力パラメータとリターン値に使用されることを示しています。

・解説

機能の解説より詳細な解説を示します。

・プログラム例

このIOCS コールを呼び出す際のシーケンスの様子、あるいはIOCS コールを使った小さなサブルーチン、プログラムなどを、例として示しています。

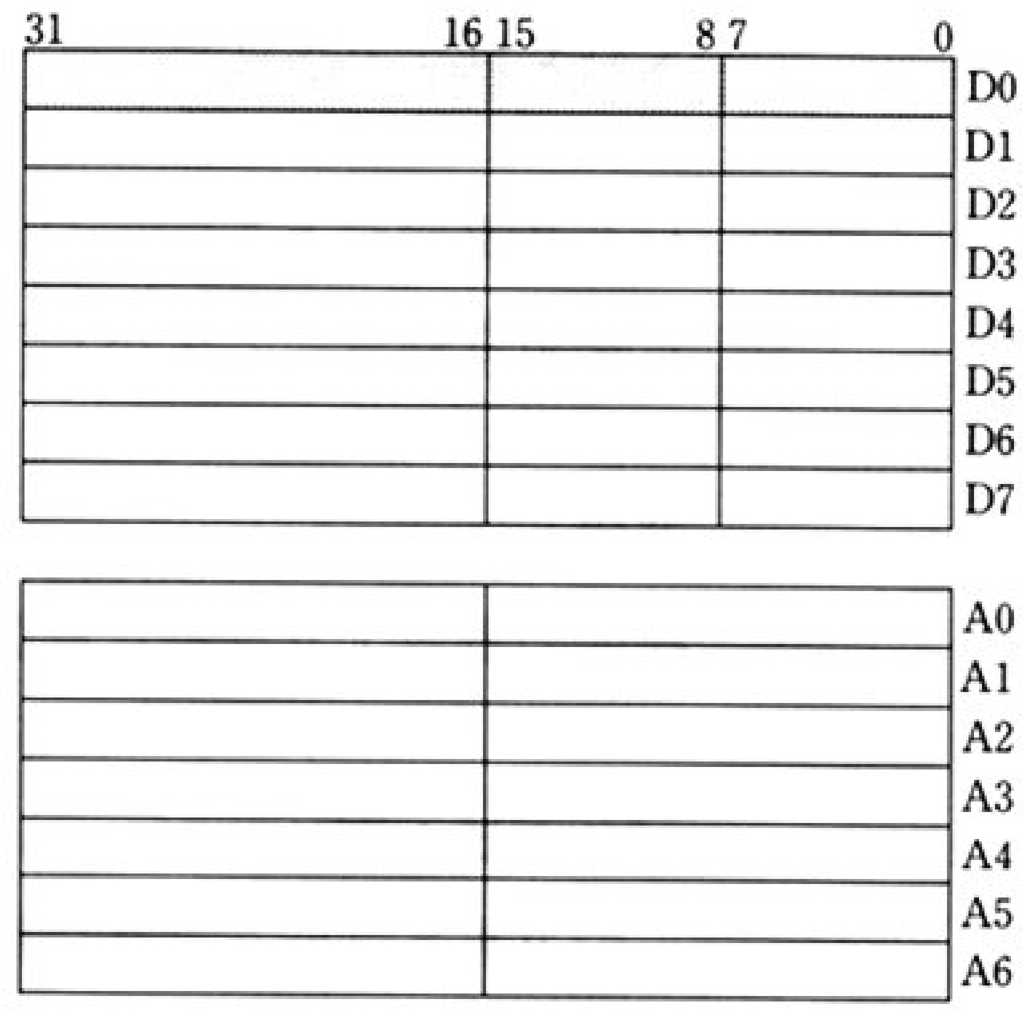
3.2.3 個々のIOCS コール

IOCS コール番号とそれぞれの機能およびその入力（パラメータ）、リターン（値）を以下に示します。
なお、次のIOCS コールはシステム予約のIOCS コールです。

IOCS コール番号：$03
IOCS コール番号：$05～$0B
IOCS コール番号：$14
IOCS コール番号：$36～$3A
IOCS コール番号：$48
IOCS コール番号：$4A
IOCS コール番号：$91～$93
IOCS コール番号：$95～$9F
IOCS コール番号：$A6～$AD
IOCS コール番号：$B0
IOCS コール番号：$D0～$D2
IOCS コール番号：$D5
IOCS コール番号：$D9～$DE
IOCS コール番号：$E0～$FF

B_KEYINP

● B_KEYINP　　IOCS コール番号：$00



機　能	キーデータの読み込み
入　力	D0. L　　$00　　IOCS コール番号
リターン	D0. L　　キーデータ

解　説

キーデータを読み込みます。

リターンの D0. L は、キーデータ（スキャンコード×256＋内部コード）です。

スキャンコードはキーの物理番号、内部コードはキーに対応する ASCII 半角コードです。

キーの物理番号とは、BITSNS（$04）で指定するキーコードグループを 8 倍したものに、そのグループのキーのビット位置の番号を加えたものです。

たとえば、‘A’が押されていたならば、グループは 3、ビット位置は 6 なので、3×8＋6＝$1E になります。

Human68k の電卓機能にも対応しています。

●電卓機能の起動方法

コマンドモードからは次の 2 通りの方法で起動できます。

①OPT. 1とOPT. 2を同時に押すと、カーソルの下の行に電卓の表示窓が現れます。

②マウスの右ボタンを 2 回押してソフトキーボードを起動してから、ソフトキーボードの右上にある電卓を左ボタンでクリックすると、ソフトキーボード内に電卓の表示窓が現れます。

ビジュアルシェルから電卓を起動するときは、文房具のアイコンを右ボタンでクリックして、ソフトキーボードを起動し、電卓を左ボタンでクリックします。

プログラム例 B_KEYINP ($00) を使ってキーボードから入力した文字を、画面に出力するプログラムです。

ESCキーを押すと終了します。

出力には、B_PUTC($20)を使用しています。

文字入力のところでは、1文字入力されるまで待ち、入力されたキーに対応する半角ASCIIコードをD0. Lにセットしています。

```
*********************************************
* B_KEYINP                              $00
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f
ESC     equ     $1b
CR      equ     $0d
LF      equ     $0a

        .text
loop:
        moveq.l #_B_KEYINP,d0   * IOCSコール番号をセット
        trap    #iocs           * d0.l <-- スキャンcode * 256 + 内部code
        tst.l   d0              * d0.l が 0 ならば
        beq     loop            *       loop へブランチ
        andi.l  #$00ff,d0       * d0.b <-- ASCII コード
        cmp.b   #ESC,d0         * d0.l が ESC ならば
        beq     exit            *       exit へブランチ
        cmp.b   #CR,d0          * d0.b が CR ならば
        bne     cr              *       CR,LF を出力
        bsr     putchr          *       putchr(CR)
        moveq.l #LF,d0          *       d0.l <-- 'LF'

cr:     bsr     putchr          * d0.l を画面に出力
        bra     loop            * goto loop


exit:   moveq.l #CR,d0          * putchr(CR)
        bsr     putchr
        moveq.l #LF,d0          * putchr(LF)
        bsr     putchr
        dc.w    _EXIT           * End of Process

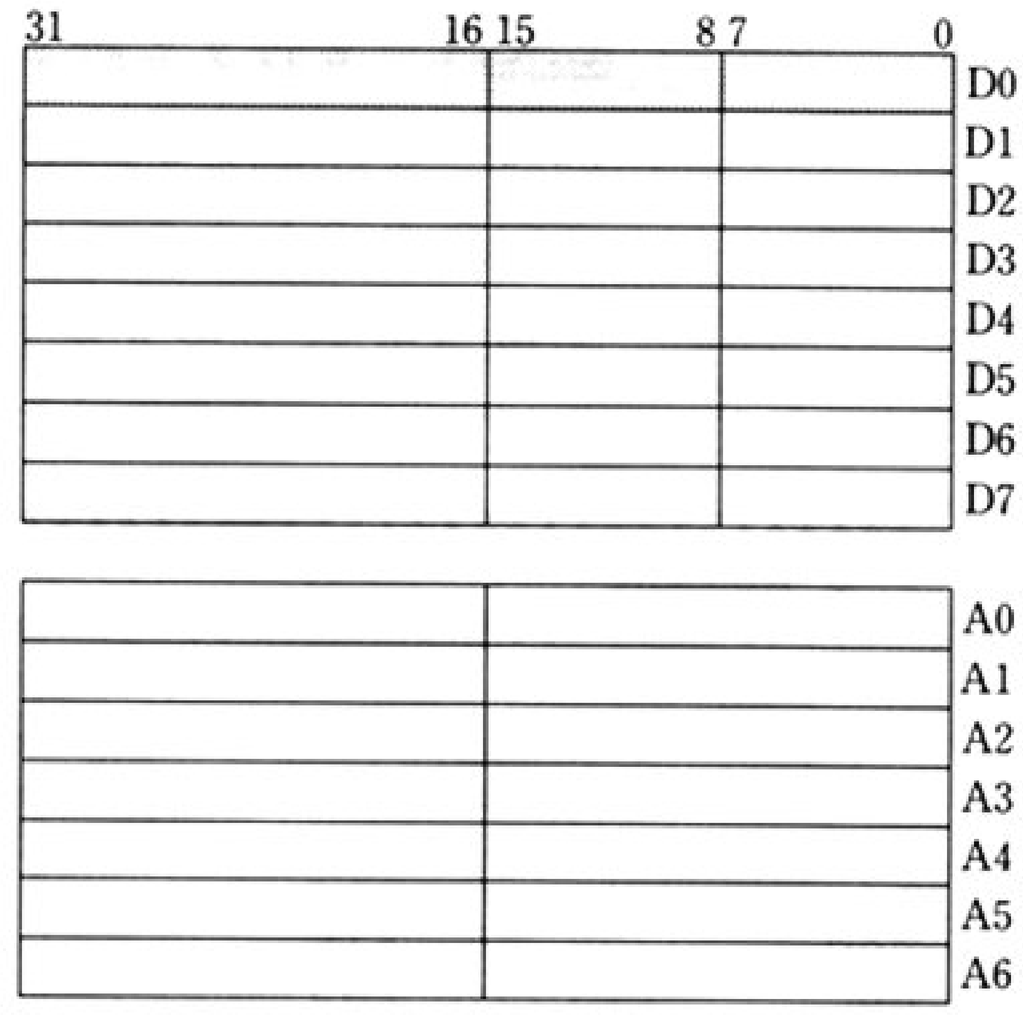
*-------------------------------------------
** PUTCHR
** < in > d0.w : 内部コード
*-------------------------------------------
putchr:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        movem.l d0-d1,-(sp)     * d0, d1 退避

        move.w  d0,d1           * 表示文字セット
        moveq.l #_B_PUTC,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール

        movem.l (sp)+,d0-d1     * d0 復帰
        rtr                     * 呼び出し元に帰ります
        .end
```

B_KEYSNS

● B_KEYSNS　　　IOCS コール番号：$01



機　能　キーの先行入力のチェック

入　力　D0. L　　$01　　IOCS コール番号

リターン　D0. L　　キーデータバッファの状態

D0. L ＝ 0 ……… 先行入力なし

D0. L ＝ $1???? … 先行入力あり ($????は、キーデータ)

解　説　キーデータバッファの状態をチェックします。

リターンのD0. Lは、$10000＋キーデータ（スキャンコード×256＋内部コード）です。

スキャンコードはキーの物理番号、内部コードはキーに対応するASCII半角コードです。

キーデータバッファの内容は変化しません。

キー入力の有無と、次にキーバッファから読み込まれるデータの種類（半角、漢字、コントロールコードなど）を調べてから各処理に分岐するときなどに使用すると、理解しやすい構造のプログラムが書けます。

Human68kの電卓機能にも対応しています。

プログラム例　キーバッファを調べるプログラムです。

```
****************************************
KEYSNS                              $01
****************************************

    .include        iocscall.mac
    .include        doscall.mac

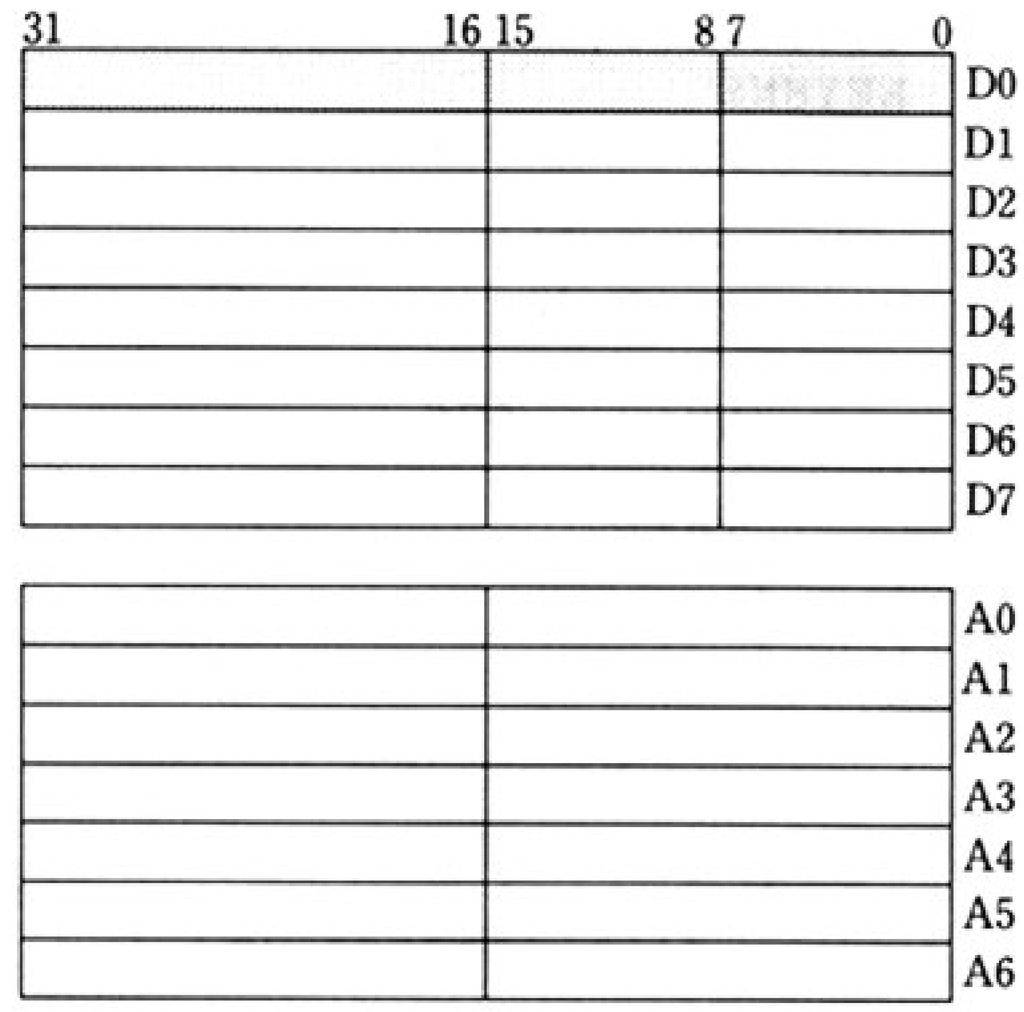
    equu    $0f

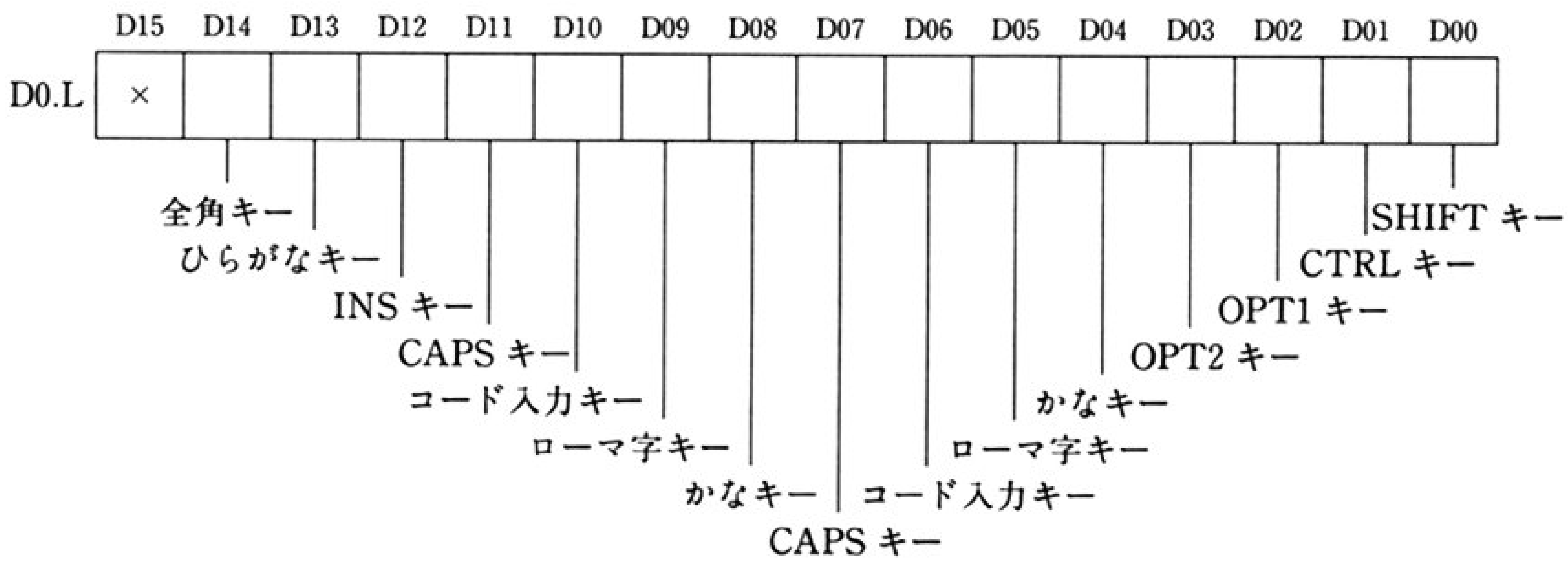
    .text
    .
    .
:
    moveq.l #_B_KEYSNS,d0    * IOCSコール番号をセット
    trap    #iocs            * d0.l <-- $0sccaa
    tst.l   d0               * d0.l が 0 ならば
    beq     loop             *        loop へブランチ
    andi.l  #$00ff,d0        * d0.b <-- ASCII コード
    .
    .
    .end
```

B_SFTSNS

● B_SFTSNS　　IOCS コール番号：$02

機　能	シフトキーの押下状態のセンス
入　力	D0. L　$02　IOCS コール番号
リターン	D0. L　シフトキーの押下状態
解　説	各シフトキーの押下状態を調べます。 D0. L の各ビットは、次のようにシフトキーに対応しています。 ビット＝1 ならシフトしています（そのビットに対応しているシフトキーが押されています）。





ここで、D00～D07 はキーが実際に押下されているかどうかを表しています。
D08～D14 は LED の点灯／消灯状態を表しています。

プログラム例　シフトキーが押されているかどうか調べるサブルーチンです。

```
********************************************
* B_SFTSNS                              $02
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equu    $0f

        .text
shiftkey:
        move.l  d0,-(sp)
        moveq.l #_B_SFTSNS,d0   * IOCSコール番号をセット
        trap    #iocs           * d0.l  0: シフトキーが押されていません
                                *       1: シフトキーが押されています
        andi.l  #1,d0           * SHIFT KEY のビット以外をリセット
        move.l  (sp)+,d0
        rts
        .end
```

KEY_INIT

● KEY_INIT　　　IOCS コール番号：$03

機　　能　キー関係の初期化（キーボードバッファクリア、及びキーボードインターフェイス初期化）

31–16	15–8	7–0	
			D0
			D1
			D2
			D3
			D4
			D5
			D6
			D7

31–16	15–0	
		A0
		A1
		A2
		A3
		A4
		A5
		A6

入　　力　D0. L　　$03　　IOCS コール番号
D1. B　　LED キーモード初期値
（付録SRAMの内容参照）

リターン　なし（D0. L は内容が保証されません）

解　　説　キーボードのバッファをクリアします。
LED キーのモードを初期化します。
この IOCS コールは、ROM モニタが起動した時に使用します。
なお、OS の CON デバイスの管理情報や、SRAM のデータエリアなどに影響を与えますので、一般のアプリケーションで使用してはいけません。

プログラム例　キー関係を初期化します。(LED キーのモードは、変更しない)

```
********************************************
* KEY_INIT                              $03
********************************************

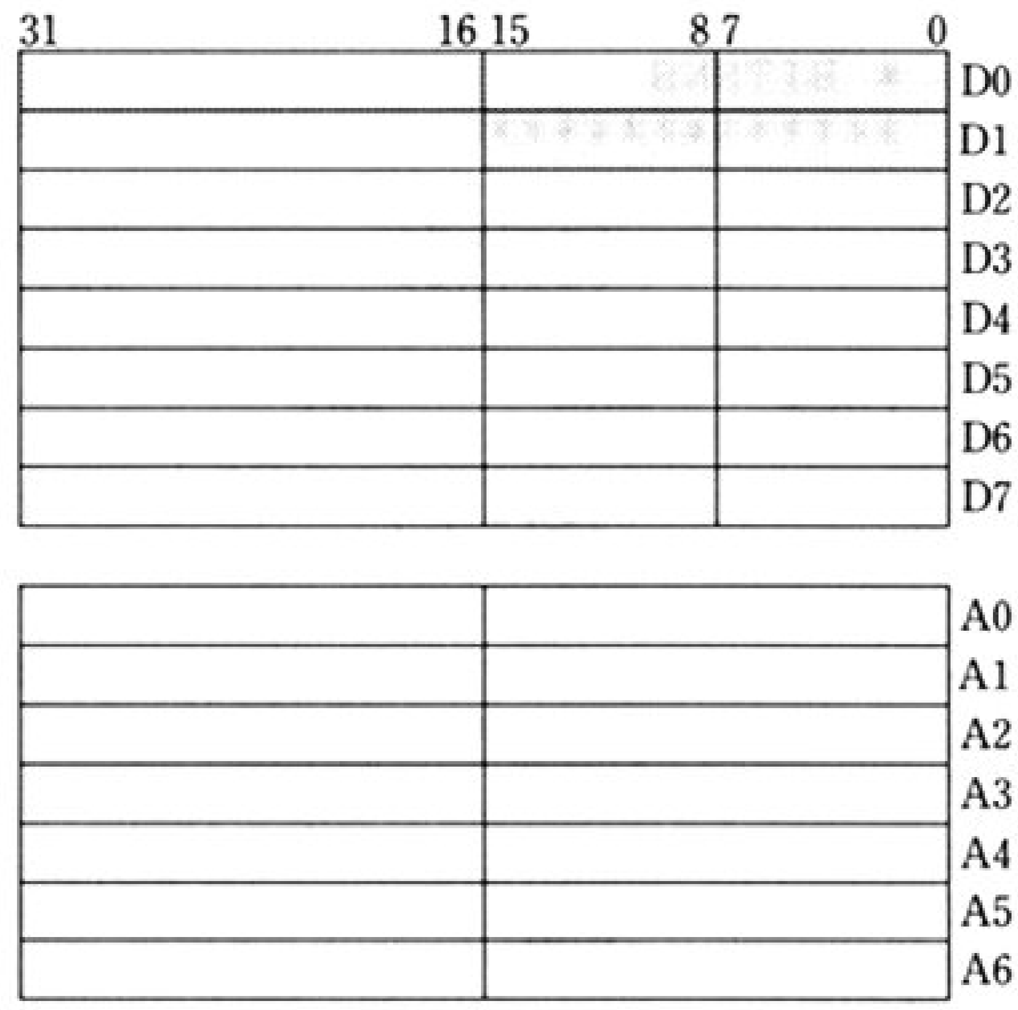
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_SFTSNS,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        lsr.w   #8,d0
        andi.w  #$007f,d0
        move.b  d0,d1           * LEDキーモード
        movea.l #_KEY_INIT,d0   * IOCSコール番号をセット
        trap    #icos           * IOCSコール
        .
        .
        .end
```

BITSNS

● BITSNS　　IOCSコール番号：$04



機　　能　キーの押下状態のセンス

入　　力　D0. L　$04　IOCSコール番号
D1. W　キーコードグループ（$00～$0F）

リターン　D0. L　キーの押下状態

解　　説　キーの押下状態を検査します。
入力のD1. Wで指定されたキーコードグループ($00～$0F)に含まれるキーの機械的な押下状態を、ビット単位でD0. Lに返します。
リターンのD0. Lは、ビット0から7が意味を持ちます（残りのビット8から31は不定です）。ビット0から7は、指定されたキーコードグループに含まれるキーの押下状態を表しています（対応するビットが1ならば、そのキーが押されていることを示します）。
キーコードグループとキー入力状態ビットとの関係を次に示します。

キーコードグループ	キー入力状態ビット							
	0	1	2	3	4	5	6	7
0		ESC	1!	2"	3#	4$	5%	6&
1	7'	8(	9)	0	−=	^	¥	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	CR	A	S
4	D	F	G	H	J	K	L	;
5	:	]	Z	X	C	V	B	N
6	M	,<	.>	/?	_	SP	HOME	DEL
7	ROLLup	ROLLdw	UNDO	←	↑	→	↓	CLR
8	／	*	−	7	8	9	+	4
9	5	6	=	1	2	3	ENTER	0
A	,	.	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F･1	F･2	F･3	F･4	F･5
D	F･6	F･7	F･8	F･9	F･10			
E	SHIFT	CTRL	OPT. 1	OPT. 2				
F								

プログラム例

```
******************************************
* BITSNS                              $04
******************************************

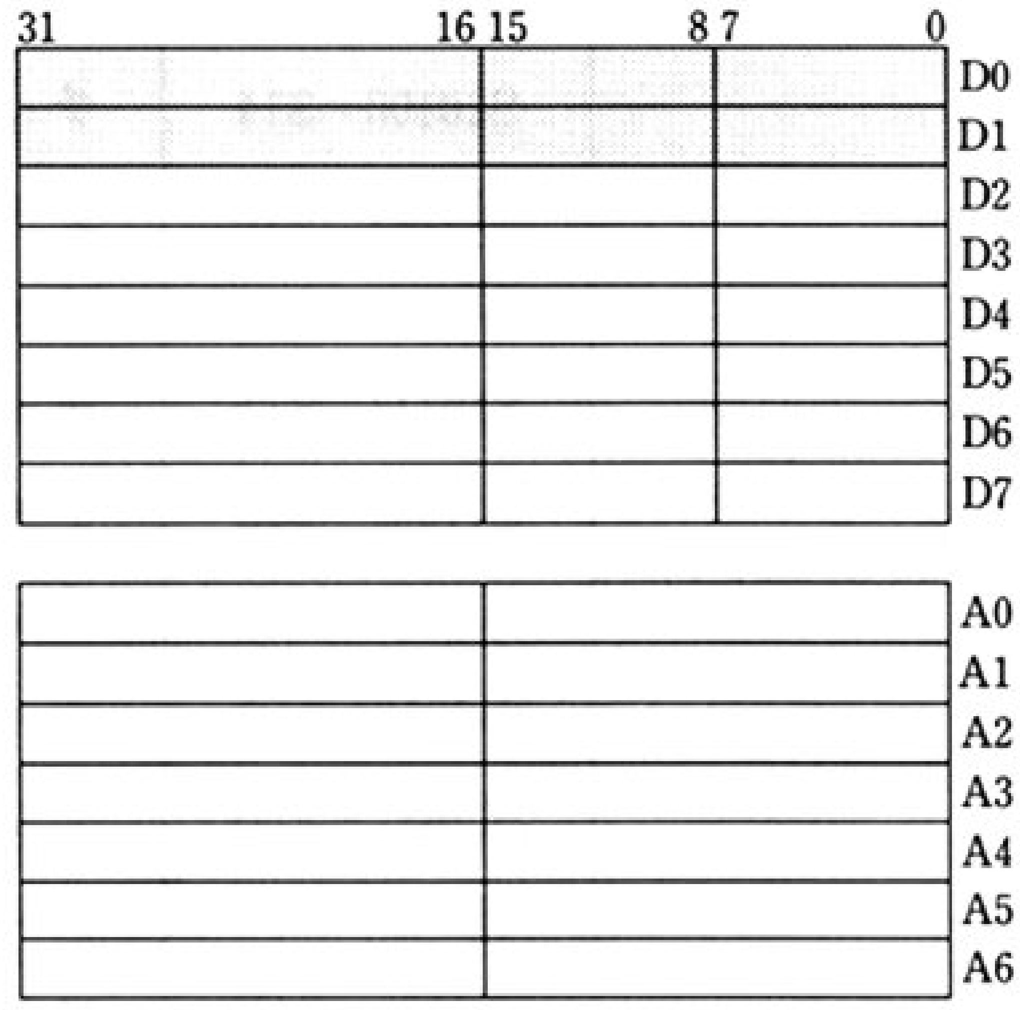
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_BITSNS,d0   * IOCSコール番号をセット
        move.w  #0,d1           * d1.w <-- キーコードグループ
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

TVCTRL

● TVCTRL　　IOCS コール番号：$0C



機　能　テレビコントロール

入　力　D0. L　$0C　IOCS コール番号
D1. L　テレビコントロールコード

リターン　なし（D0. L は内容が保証されません）

解　説　ディスプレイの制御（テレビコントロール）を行います。
入力の D0. L で指定される値は、次に示す通りです。

ビット 543210	コントロールコード
S00001＝$01	ボリュームを上げる
S00010＝$02	ボリュームを下げる
S00011＝$03	ボリュームを標準にする
S00100＝$04	チャンネルコール
S00101＝$05	テレビ画面（初期化・リセット）
S00110＝$06	音声ミュート
S00111＝$07	電源 ON
S01000＝$08	テレビ／コンピュータ
S01001＝$09	テレビ／外部、コンピュータ標準／オーバー
S01010＝$0A	コントラスト標準
S01011＝$0B	チャンネルアップ
S01100＝$0C	チャンネルダウン
S01101＝$0D	電源 OFF
S01110＝$0E	電源 ON／OFF
S01111＝$0F	スーパーインポーズ 1
S10000＝$10	チャンネル 1
S10001＝$11	チャンネル 2
S10010＝$12	チャンネル 3
S10011＝$13	チャンネル 4

ビット 543210	コントロールコード
S10100=$14	チャンネル 5
S10101=$15	チャンネル 6
S10110=$16	チャンネル 7
S10111=$17	チャンネル 8
S11000=$18	チャンネル 9
S11001=$19	チャンネル 10
S11010=$1A	チャンネル 11
S11011=$1B	チャンネル 12
S11100=$1C	テレビ画面 ($05)
S11101=$1D	コンピュータ画面 ($05+$08)
S11110=$1E	スーパーインポーズ 1 ($05+$0F)
S11111=$1F	スーパーインポーズ 2 ($05+$0F+$0A)

注：Sビット(ビット5)が1 (1????? ($20+コントロールコード)：$21~$3F)のとき、テレビの電源をONしたのち、コントロールコードを実行します。

プログラム例 テレビ音声をミュートします。

```
********************************************
* TVCTRL                                 $0C
********************************************

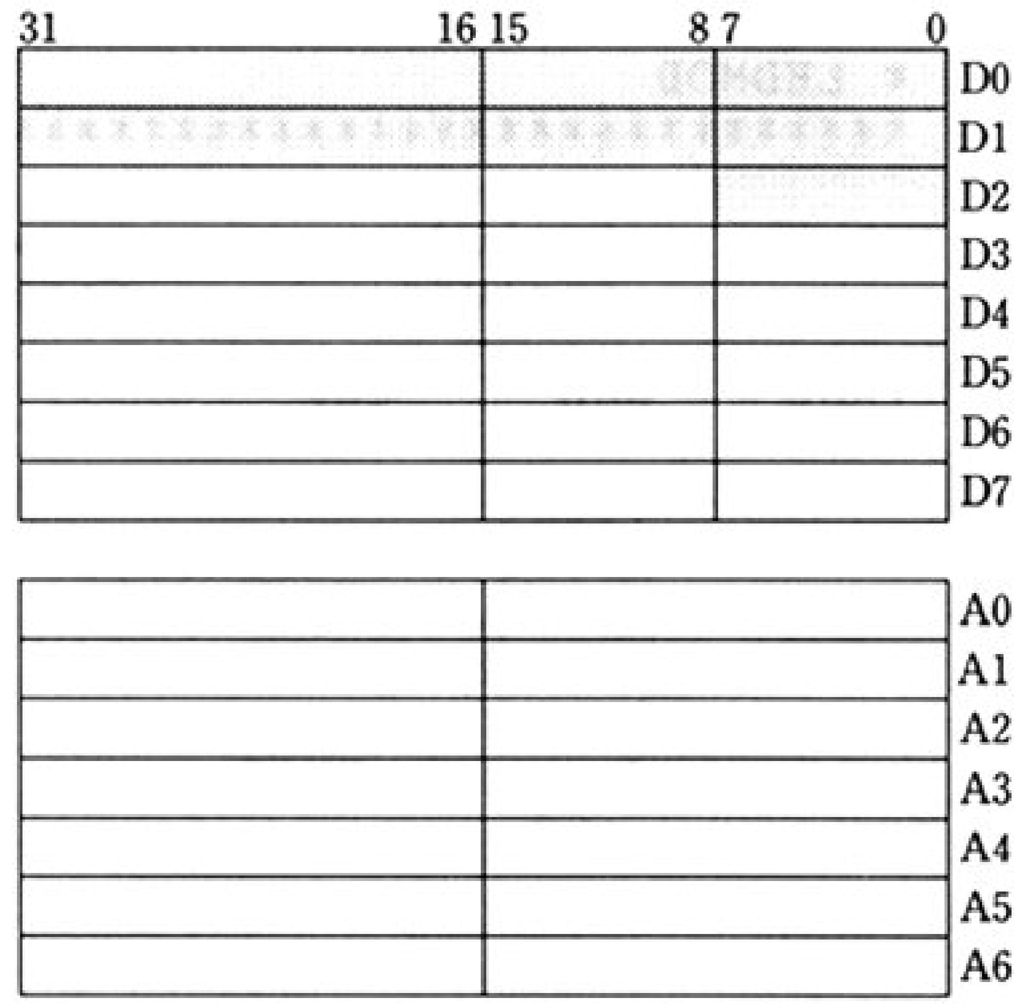
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
mute:
        moveq.l #_TVCTRL,d0     * IOCSコール番号をセット
        moveq.l #$06,d1         * d1.l <-- 音声ミュート
        trap    #iocs           * d0.l <-- 上の表に従ったデータ
        dc.w    _EXIT           * プロセス終了システムコール
        .end
```

LEDMOD

● LEDMOD　　IOCSコール番号：$0D



機　能　LEDキーモード設定

入　力　D0. L　$0D　IOCSコール番号
D1. L　LEDキーの番号
D2. B　指定キーのモードのオン／オフ

リターン　なし（D0. Lは内容が保証されません）

解　説　LEDキーのモードを設定します。
入力のD1. LでLEDキーの種類を指定します。
D2. BでLEDキーをオンにするかオフにするか（LEDも点灯または消灯）を指定します。
D2. Bの内容は次の通りです。

D2. B ＝ 0……指定キーのモードをオフ
D2. B ＝ 1……指定キーのモードをオン

LEDのキーの番号の内容を次に示します。

D1. L	LEDキー
0	かな
1	ローマ字
2	コード入力
3	CAPS
4	INS
5	ひらがな
6	全角

プログラム例

```
*******************************************
* LEDMOD                              $0D
*******************************************

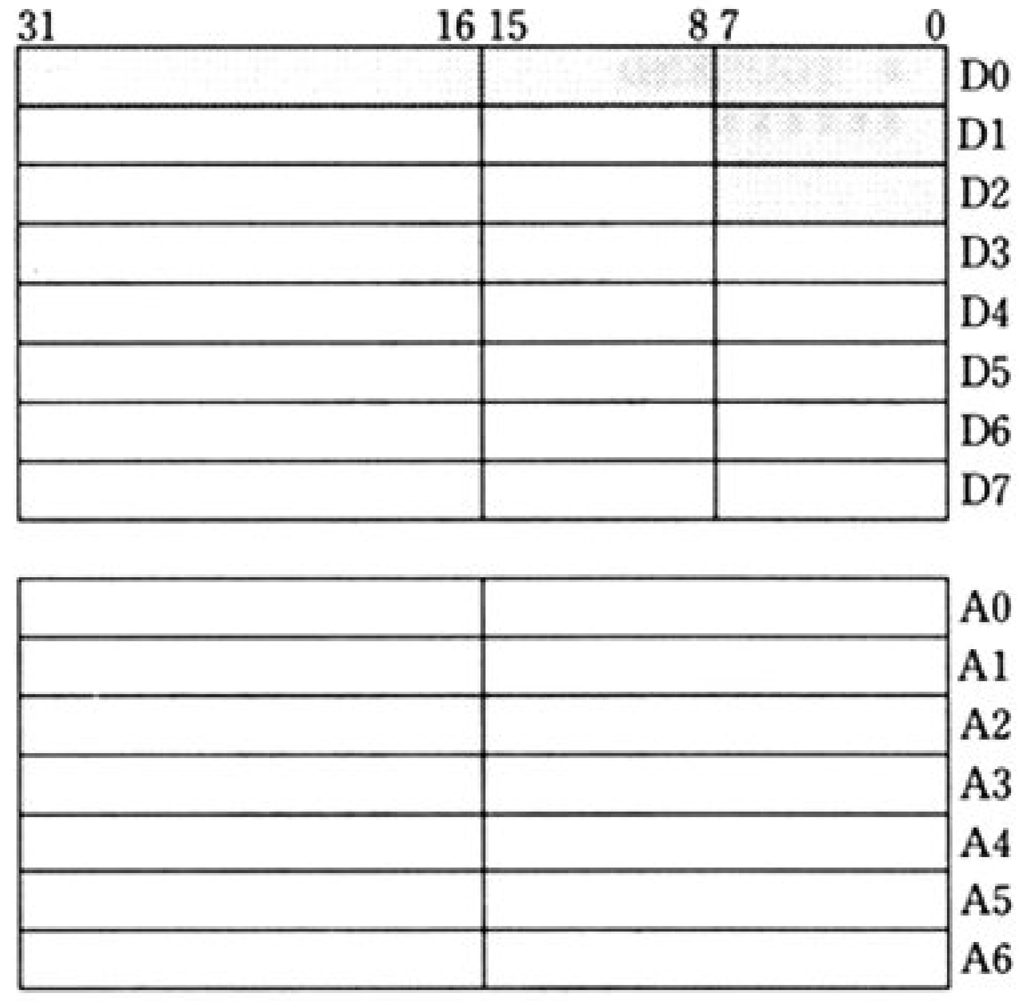
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        moveq.l #_LEDMOD,d0     * IOCSコール番号をセット
        moveq.l #1,d1           * ローマ字キー
        move.b  #1,d2           * ON
        trap    #iocs           * IOCSコール
        dc.w    _EXIT           * プロセス終了システムコール
        .end
```

TGUSEMD

● TGUSEMD　　IOCS コール番号：$0E



機　　能　テキスト／グラフィック画面の使用状態設定

入　　力
D0. L　　$0E　IOCS コール番号
D1. B　　グラフィック／テキスト
D2. B　　画面の使用状態

リターン　D0. L　　画面のモード

解　　説　テキスト／グラフィック画面の使用状態を設定します。

入力の D1. B はグラフィック画面、またはテキスト画面を指定します。

D1. B ＝ 0……グラフィック画面（$C00000～$DFFFFF)
D1. B ＝ 1……テキスト画面（$E40000～$E7FFFF)

D2. B では使用状態を指定します。

D2. B ＝ 0……誰も使用していない
D2. B ＝ 1……システムで使用している（ソフトキーボード／電卓／RAM ディスク）
D2. B ＝ 2……アプリケーションで使用している
D2. B ＝ 3……アプリケーションで使用したあとで壊れたままです
D2. B ＝ —1…どのモードかを読み取る

リターンの D0. L は、画面のモードを示します（0～3：入力の D2. B と同じ）

プログラム例

```
*******************************************
* TGUSEMD                              $0E
*******************************************

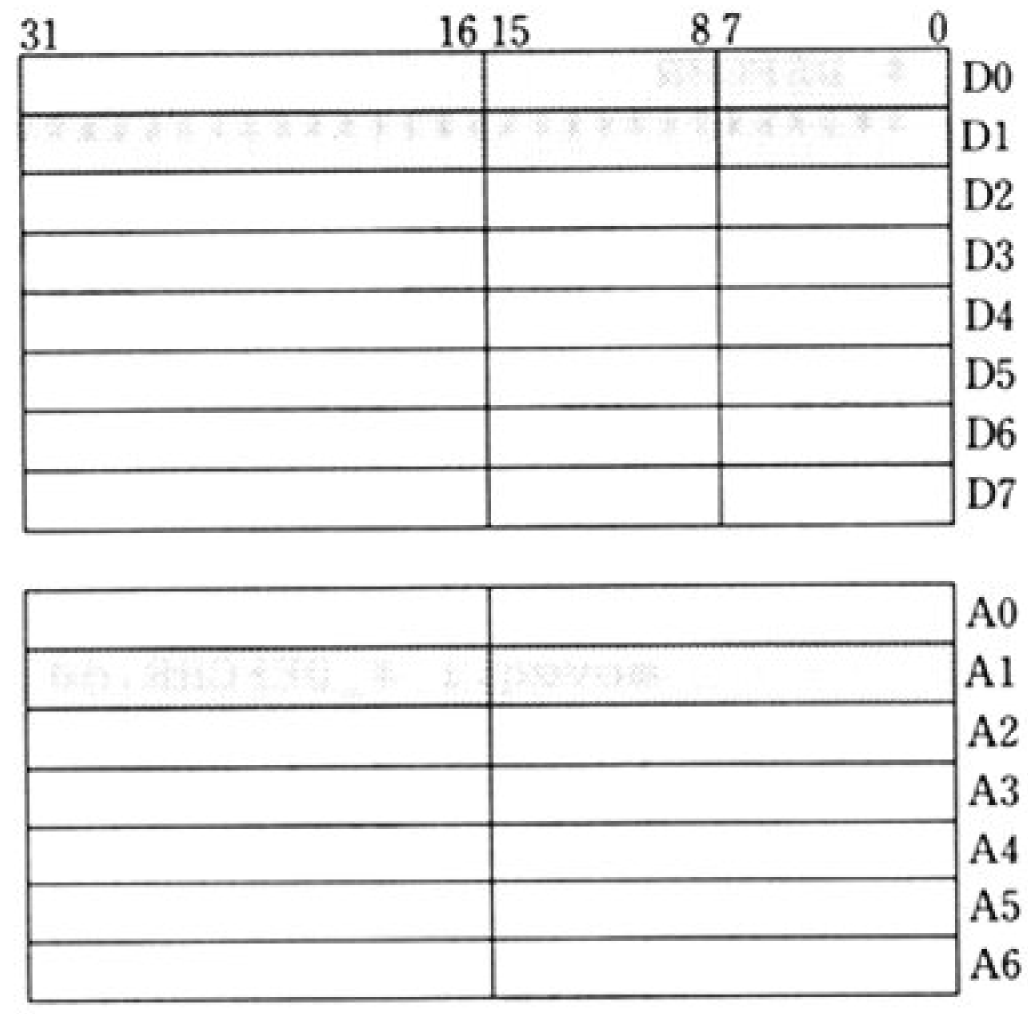
        .include        iocscall.mac
        .include        doscall.mac

iocs    equu     $0f

        .text
        .
        .
        moveq.l #_TGUSEMD,d0    * IOCSコール番号をセット
        move.b  #0,d1           * テキスト画面
        move.b  #3,d2
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

DEFCHR

● DEFCHR　　　IOCS コール番号：$0F



機　　能　漢字パターンの外字エリア書き込み

入　　力
D0. L　　$0F　IOCS コール番号
D1. L　　漢字コード
A1. L　　外字パターンの先頭アドレス
　　　　（先頭からデータ）
(A1). B　パターンデータ

リターン　D0. L　　終了コード
D0. L ＝ 0………正常終了
D0. L ＝ −1……異常終了（外字でない）

解　　説
指定の漢字パターンを、外字エリアに書き込みます。
入力の D1. L は、漢字パターンの文字の大きさと、格納する漢字コードを指定します。
上位 16 ビットは、漢字パターンの文字の大きさを指定します。
内容は次の通りです。

0, 8……16×16、8×16 ドット
12　……24×24、12×24 ドット

下位 16 ビットは、漢字パターンを格納する漢字コード（シフト JIS コード：JIS 漢字コード（$7621～$777E）も可）を指定します。
入力の A1. L で漢字パターンの先頭アドレスを指定します。
入力の(A1). B でパターンデータを指定します。
スーパーバイザ領域のアドレスも指定できますが、不正なアドレスを指定しないように注意してください。

プログラム例

```
*******************************************
* DEFCHR                                $0F
*******************************************

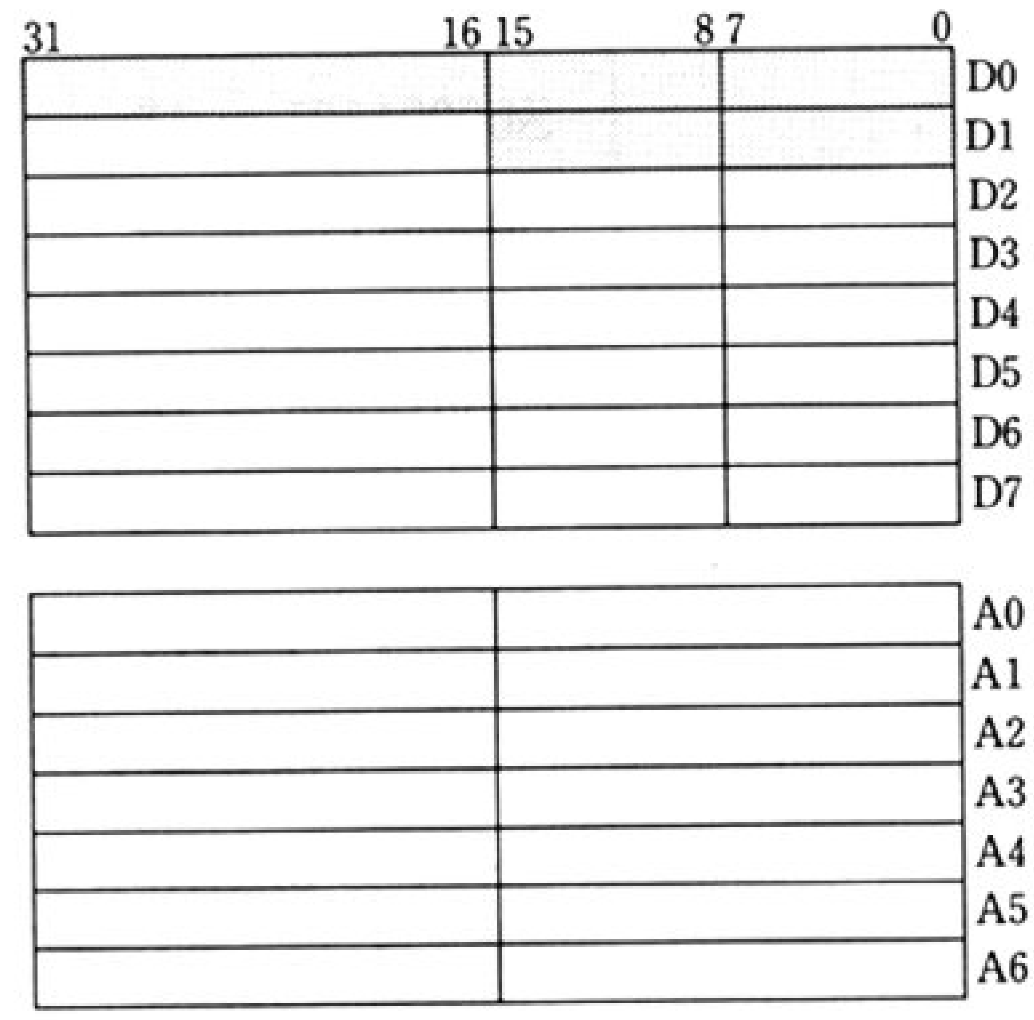
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DEFCHR,d0     * IOCSコール番号をセット
        move.l  #$eaa1,d1       * $eaa1 はシフトJISコード
        lea     font,a1         * パターン先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
font:
        dc.w    %0000000000100000
        dc.w    %0110000001111110
        dc.w    %0011000110000100
        dc.w    %0001011001001000
        dc.w    %0000000100110000
        dc.w    %0000000011000000
        dc.w    %0000011100000000
        dc.w    %1111000111111100
        dc.w    %0001001100100000
        dc.w    %0001000000100000
        dc.w    %0001011111111111
        dc.w    %0001000000100000
        dc.w    %0001001000100010
        dc.w    %0011001111111110
        dc.w    %0110100000000000
        dc.w    %1100011111111111
        .
        .
        .end
```

CRTMOD

● CRTMOD　　IOCS コール番号：$10



機　　能　CRT のモード設定

入　　力　D0. L　$10　IOCS コール番号
D1. W　CRT モード

リターン　D1. W ＝ −1：
D0. L　現在の CRT モード
D1. W ≠ −1：
なし (D0. L の内容は保証されません)

解　　説　CRT のモードを指定します。
テキスト画面をクリアして、表示モードにします。
テキストパレットは標準に戻ります。
グラフィック画面はクリアせず、無表示モードにします。
スプライト・BG もクリアせず、無表示モードにします。
入力の D1. W は、CRT モードを指定します。
内容を次に示します。

ビット 876543210	CRT モード	表示画面	表示色	実画面
S00000000 = 0	高解像度	512×512	16／16 色	1024
S00000001 = 1	標準	512×512	16／16 色	1024
S00000010 = 2	高解像度	256×256	16／16 色	1024
S00000011 = 3	標準	256×256	16／16 色	1024
S00000100 = 4	高解像度	512×512	16／16 色	512
S00000101 = 5	標準	512×512	16／16 色	512
S00000110 = 6	高解像度	256×256	16／16 色	512
S00000111 = 7	標準	256×256	16／16 色	512
S00001000 = 8	高解像度	512×512	16／256 色	512
S00001001 = 9	標準	512×512	16／256 色	512
S00001010 = 10	高解像度	256×256	16／256 色	512
S00001011 = 11	標準	256×256	16／256 色	512
S00001100 = 12	高解像度	512×512	16／65536 色	512

ビット 876543210	CRT モード	表示画面	表示色	実画面
S00001101 = 13	標準	512×512	16／65536 色	512
S00001110 = 14	高解像度	256×256	16／65536 色	512
S00001111 = 15	標準	256×256	16／65536 色	512
S00010000 = 16	高解像度	768×512	16／16 色	1024
=−1	現在のモードを返す			

注：Sビット（ビット8）が1（01?? ($100+CRT モード)：$00〜$10）のとき、モード切り換えだけを行い、画面クリア、パレット／コントラスト／表示モードの初期化はしません。

D0. L は、D1. W ＝ −1のとき現在のモードを返します。

グラフィック画面の表示は G_CLR_ON（$90）で行います。

プログラム例 CRT のモードを 768×512 ドット、16 色に設定します。

```
*******************************************
* CRTMOD                              $10
*******************************************

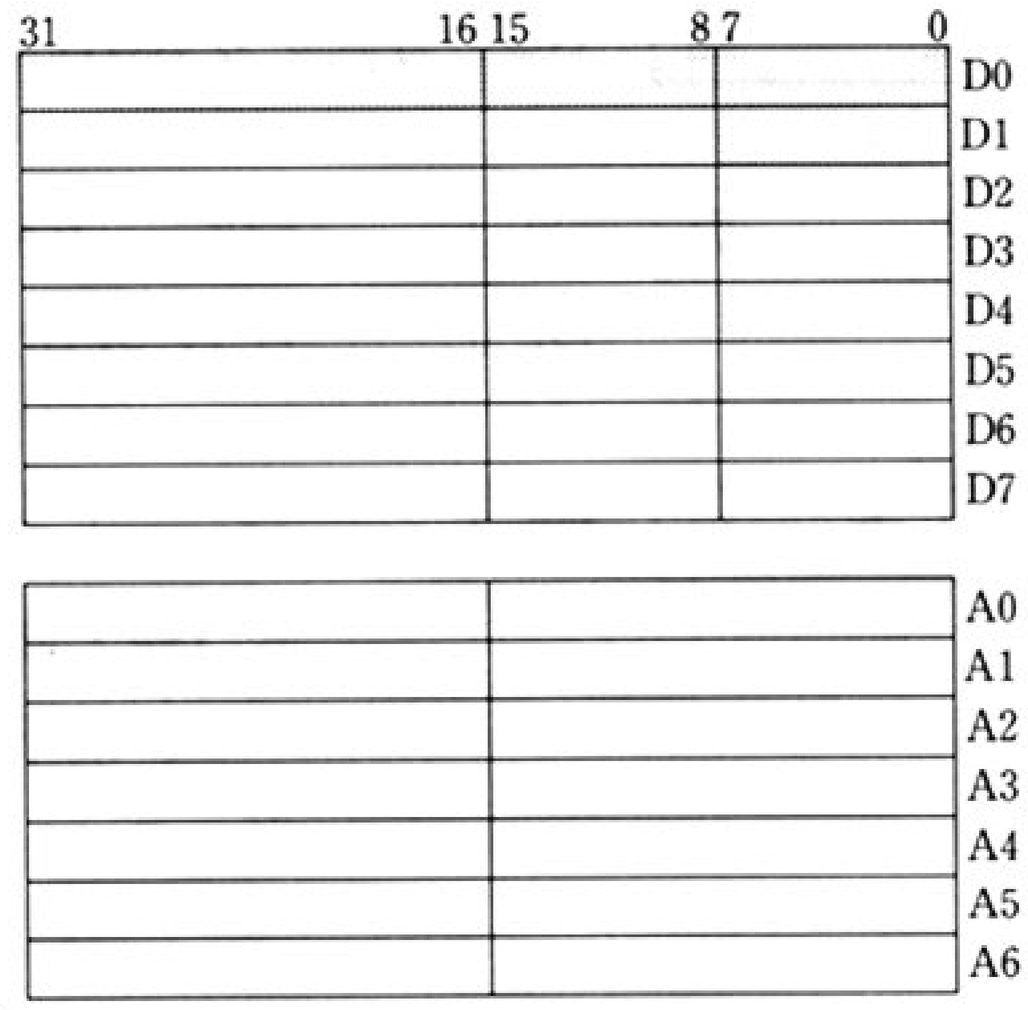
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_CRTMOD,d0     * IOCSコール番号をセット
        move.w  #16,d1          * d1.w <-- 768×512、16 色
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

CONTRAST

● CONTRAST　　IOCS コール番号：$11



機　能　コントラスト設定

入　力　D0. L　$11　IOCS コール番号
D1. B　コントラストの設定

リターン　D0. L　前のコントラスト

解　説　コントラストの設定を行います。
入力の D1. B には次の値を指定します。

D1. B ＝ 0～15……コントラストの設定
D1. B ＝ —1………コントラストの通知要求
D1. B ＝ —2………システムの既定値に戻します

D0. L には設定変更前のコントラストが返されます。

プログラム例　コントラストをシステム既定値にします。

```
********************************************
* CONTRAST                              $11
********************************************

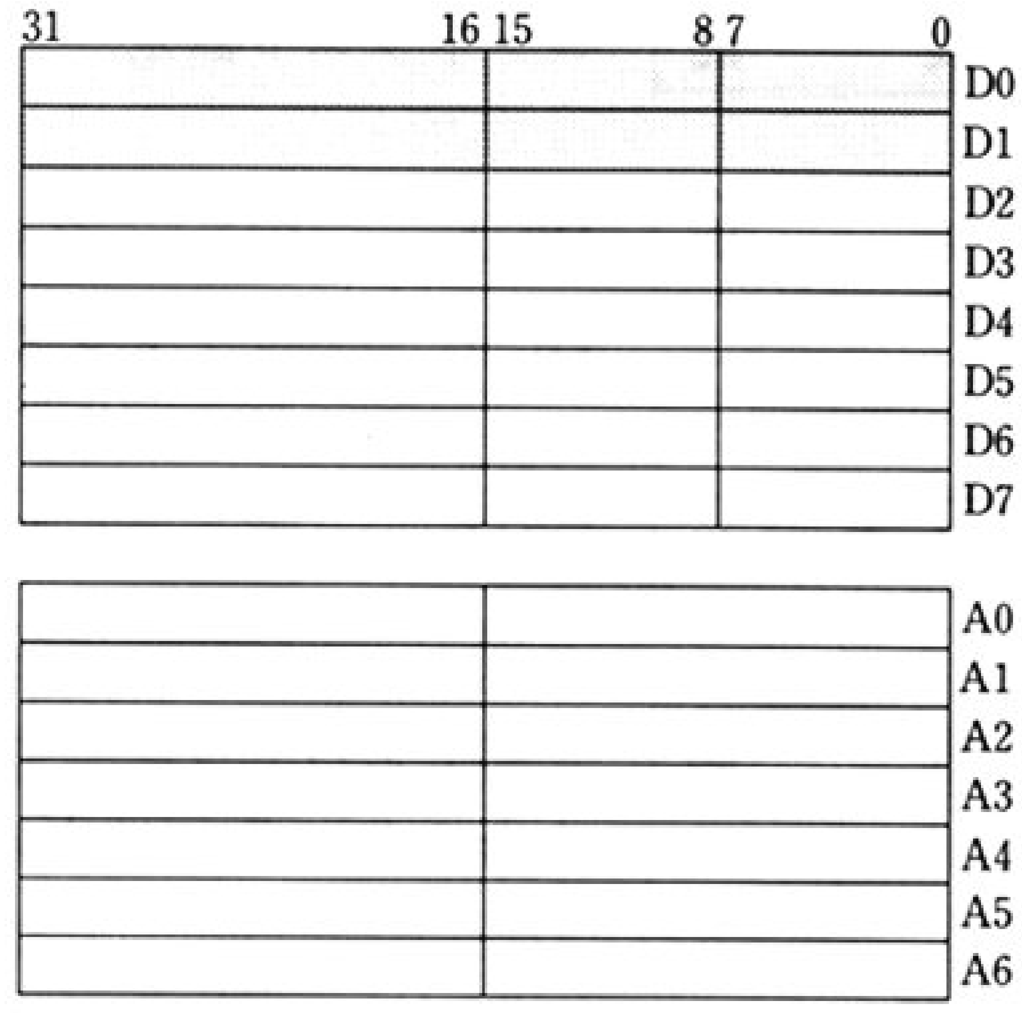
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        movem.l d0-d1,-(sp)
        moveq.l #_CONTRAST,d0     * IOCSコール番号をセット
        move.b  #-2,d1            * d1.b <-- システム既定値指定
        trap    #iocs             * IOCSコール
        movem.l (sp)+,d0-d1
        rts
        .
        .
        .end
```

HSVTORGB

● **HSVTORGB　　IOCS コール番号：$12**



機　　能　HSV 方式から RGB 方式への変換

入　　力　D0. L　　$12　　IOCS コール番号
D1. L　　HSV 方式データ

リターン　D0. L　　パレットで指定する RGB 方式（緑・赤・青、各5ビットずつ、最下位ビットは0）のデータ

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
D0. L	G	G	G	G	G	R	R	R	R	R	B	B	B	B	B	0

解　　説　HSV 方式から RGB 方式に変換します。
HSV 方式は、色相、彩度、明度による色の指定です。
入力の D1. L は、HSV 方式のデータで、次の形式で記述されます。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D1. L	0	0	0	0	0	0	0	0	I	I	I	F	F	F	F	F

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	S	S	S	S	S	0	0	0	V	V	V	V	V

ビット 23～21：　III ＝ 000～101……カラー
＝ 000 …………赤～黄色
＝ 001 …………黄色～緑
＝ 010 …………緑～シアン
＝ 011 …………シアン～青
＝ 100 …………青～マゼンダ
＝ 101 …………マゼンダ～赤
＝ 11? …………禁止
ビット 20～16：　FFFFF ＝ 0～$1F ……III の間の色合い
ビット 12～ 8：　SSSSS ＝ 0～$1F ……彩度（0で白、$1F で原色）
ビット 4～ 0：　VVVVV ＝ 0～$1F ……明度（0で黒、$1F で原色）

プログラム例　HSVで表した暗い赤からRGBデータを求めます。

```
*******************************************
* HSVTORGB                             $12
*******************************************

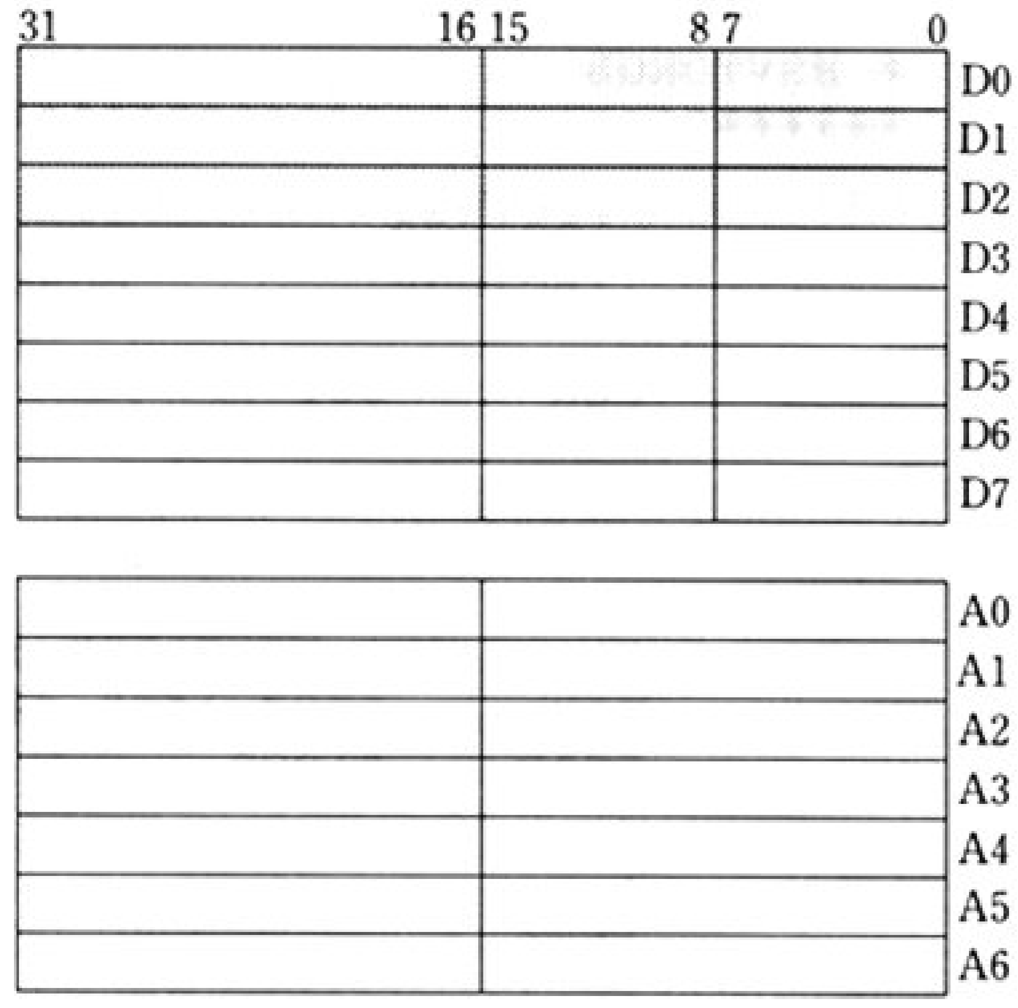
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_HSVTORGB,d0    * IOCSコール番号をセット
        move.l  #$00_1f08,d1     * $00_1f08 = %00000000_00011111_00001000
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

TPALET

● TPALET　　IOCS コール番号：$13



機　　能　テキストのパレット指定

入　　力
D0. L　　$13　　IOCS コール番号
D1. B　　パレットコード
D2. L　　カラーコード

リターン
D2. L ＝ −1：
　D0. L　　現在のカラーコード
D2. L ≠ −1：
　なし（D0. L の内容は保証されません）

解　　説
テキストパレットに、カラーコードを設定します。
入力の D1. B は、指定するテキストパレットの番号です。
テキストパレットの 0〜3 は、テキストに表示する文字の色指定に使われます。
テキストパレット 4〜7 には、同一のカラーコードが設定されます。
テキストパレット 8〜15 にも、同一のカラーコードが設定されます。
4〜7、8〜15 はソフトキーボード／電卓で使用されます。
内容は次の通りです。

D1. B ＝ 0 …………テキストカラー 0
D1. B ＝ 1 …………テキストカラー 1
D1. B ＝ 2 …………テキストカラー 2
D1. B ＝ 3 …………テキストカラー 3
D1. B ＝ 4〜7………ソフトキーボード／電卓カラー 0
　　　　　　　　4〜7 に同じカラーコードをセットします。
D1. B ＝ 8〜15 ……ソフトキーボード電卓カラー 1
　　　　　　　　8〜15 に同じカラーコードをセットします。

入力の D2. L は、指定するテキストのパレットのカラーコード(0〜$FFFF)です。
D2. L に−1 を指定した場合は現在のカラーを返し、−2 の場合は、パレットをシステムの既定値に戻します。

プログラム例

```
******************************************
* TPALET                              $13
******************************************

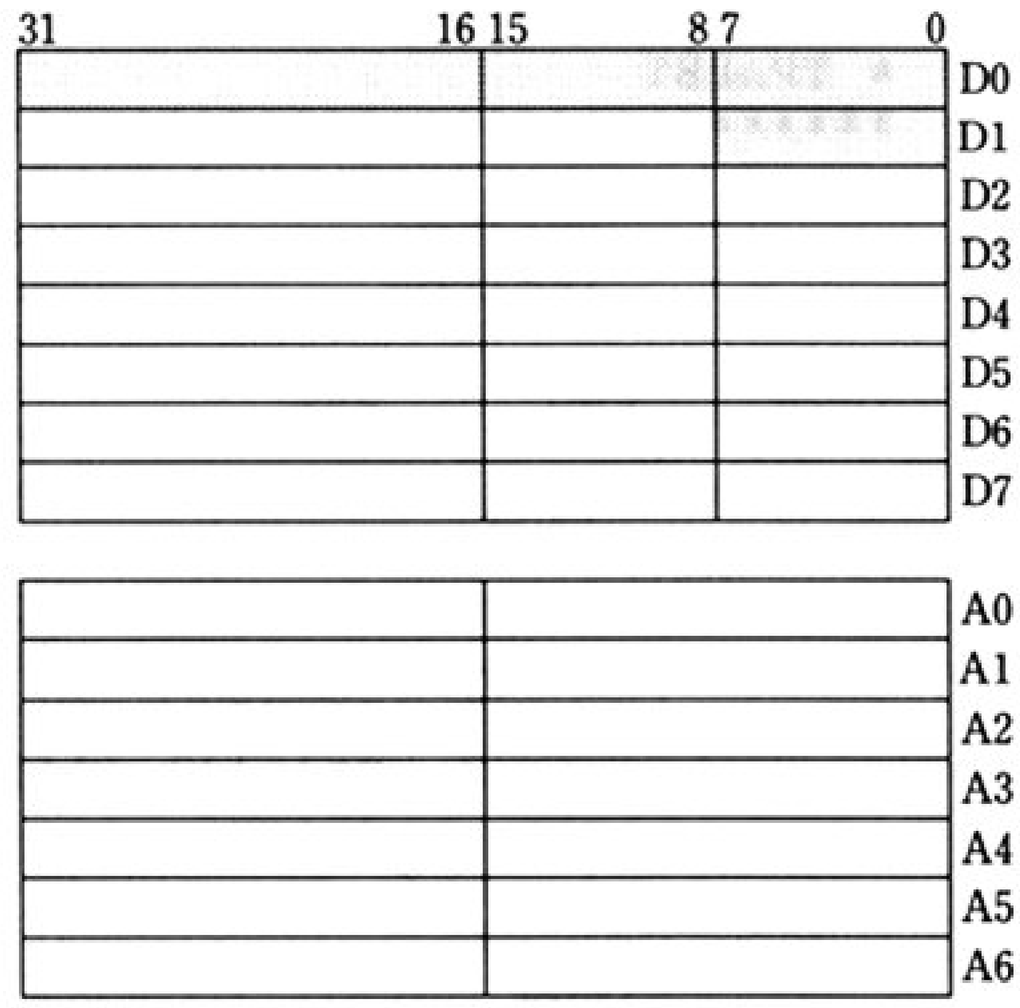
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TPALET,d0              * IOCSコール番号をセット
        move.b  #1,d1                    * パレットコード
        move.l  #%11111_11111_11111_0,d2* カラーコード
        trap    #iocs                    * IOCSコール
        .
        .
        .end
```

TCOLOR

● TCOLOR　　IOCS コール番号：$15



機　能　テキスト VRAM 指定

入　力　D0. L　$15　IOCS コール番号
D1. B　プレーン番号

リターン　なし（D0. L は内容が保証されません）

解　説　TEXTGET（$1A）、TEXTPUT（$1B）、CLIPPUT（$1C）でアクセスされるテキスト VRAM を選択します。

TEXTGET($1A)……指定のドット座標から指定アドレスへパターンを読み込む
TEXTPUT($1B)……指定アドレスのパターンを指定ドット座標に書き込む
CLIPPUT($1C)　……指定アドレスのパターンを指定ドット座標に書き込む

指定した IOCS コールを終了した後は、テキスト VRAM を 1 に戻す（D1＝1）処理を行ってください。
そうしない場合、その他の IOCS コールの動作は保証されません。
入力の D1. B は、ビット単位でテキスト VRAM のプレーン番号を指定します。
内容は次の通りです。
プレーンは単一のプレーン（1、2、4、8 のみ）を指定してください。

ビット3	ビット2	ビット1	ビット0	指定するプレーン番号
0	0	0	0	テキストプレーン 0
?	?	?	1	テキストプレーン 0
?	?	1	0	テキストプレーン 1
?	1	0	0	テキストプレーン 2
1	0	0	0	テキストプレーン 3

?は、0、1 どちらでも可。

プログラム例

```
*******************************************
* TCOLOR                              $15
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

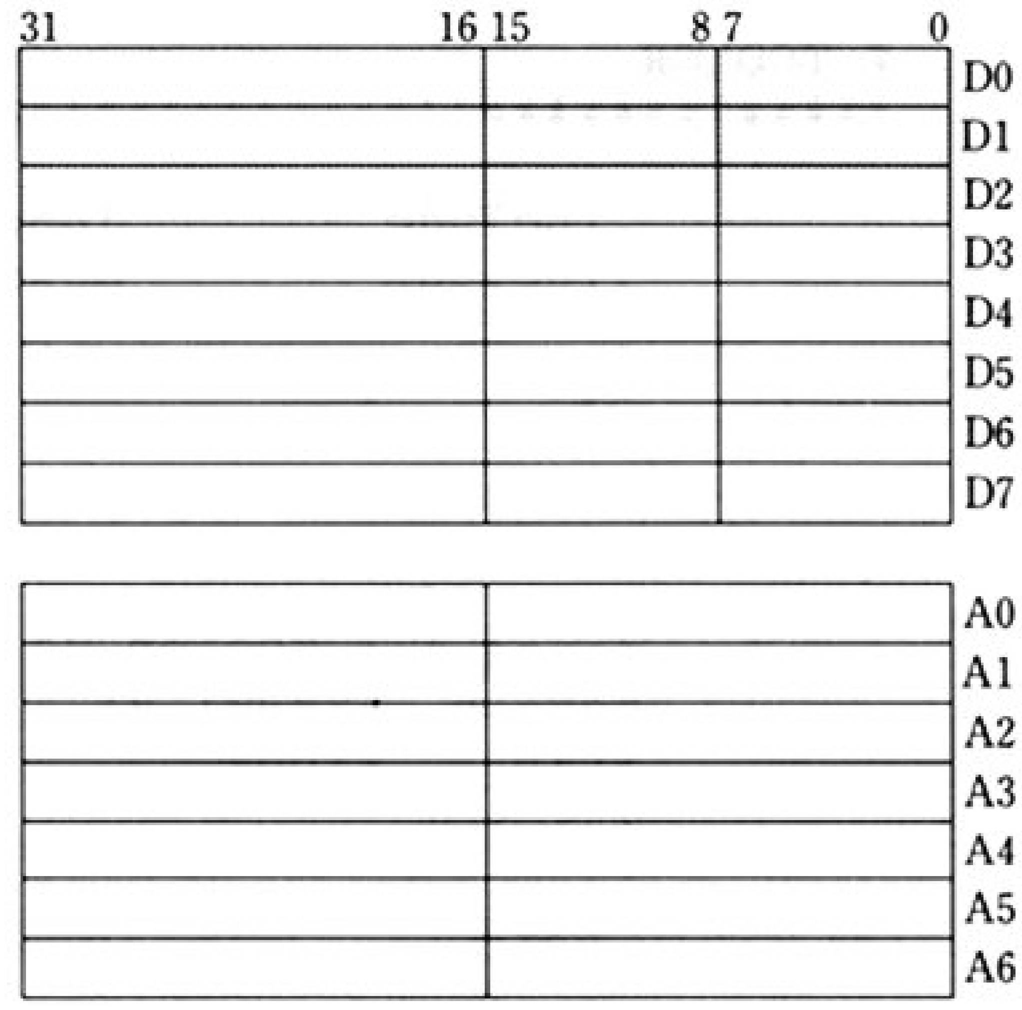
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TCOLOR,d0     * IOCSコール番号をセット
        move.b  #%0001,d1       * テキストプレーン 0 指定
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

FNTADR

● FNTADR　　IOCS コール番号：$16

機能　漢字パターンのアドレスを求める



入力

D0. L　　$ 16　　IOCS コール番号

D1. W　　シフト JIS コード

D2. L　　文字の大きさ

リターン

D0. L　　パターンのアドレス

D1. W　　（文字パターンの X 方向のバイト数）－1

D2. W　　（文字パターンの Y 方向のドット数）－1

解説

指定の漢字パターンが格納されている ROM の、アドレスを求めます。

スーパーバイザ領域のアドレスを返しますので、ユーザーモードで読み書きしないように注意してください。

入力の D1. W は指定する漢字コードを、シフト JIS コードで指定します。

指定は、JIS 漢字コード（$2121～$777E）も可能です。

入力の D2. L は指定する漢字パターンの文字の大きさで、内容は次の通りです。

D2. L ＝　6 … 12×12、6×12 ドット

D2. L ＝　8 … 16×16、8×16 ドット

D2. L ＝ 12 … 24×24、12×24 ドット

リターンは、D0. L にバッファアドレスを、D1. W に（文字パターンの X 方向のバイト数）－1 を、D2. W に（文字パターンの Y 方向のドット数）－1 をそれぞれ返します。

各フォントパターンは、次の通り。

リターンのパターンアドレスは、スーパーバイザ領域ですのでアクセスすると、アドレスエラーが発生します。

この IOCS コールはシステムで使用しますので、一般のアプリケーションでは使用しないようにしてください。

● 6×12

ビット D07　　D01 D00

フォント	0	0	+0
データ	0	0	+1
:	:		:
:	:		:
:	0	0	+10
:	0	0	+11
			バイト

● 8×16

ビット D07　　D00

フォントデータ	+0
:	+1
:	:
:	:
:	+14
:	+15
	バイト

● 12×12

ビット D15　　D03　　D00

フォント	0	0	0	0	+0
データ	0	0	0	0	+2
:	:				:
:	:				:
:	0	0	0	0	+20
:	0	0	0	0	+22
					バイト

● 12×24

ビット D15　　　D03　　D00

フォント	0 0 0 0	+0
データ	0 0 0 0	+2
:	:	:
:	:	:
:	0 0 0 0	+44
:	0 0 0 0	+46
		バイト

● 16×16

ビットD15　　　　D00

フォントデータ	+0
:	+2
:	:
:	:
:	+28
:	+30
	バイト

● 24×24

ビット D23　　　　　D00

フォントデータ	+0
:	+3
:	:
:	:
:	+66
:	+69
	バイト

プログラム例

```
*******************************************
* FNTADR                               $16
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

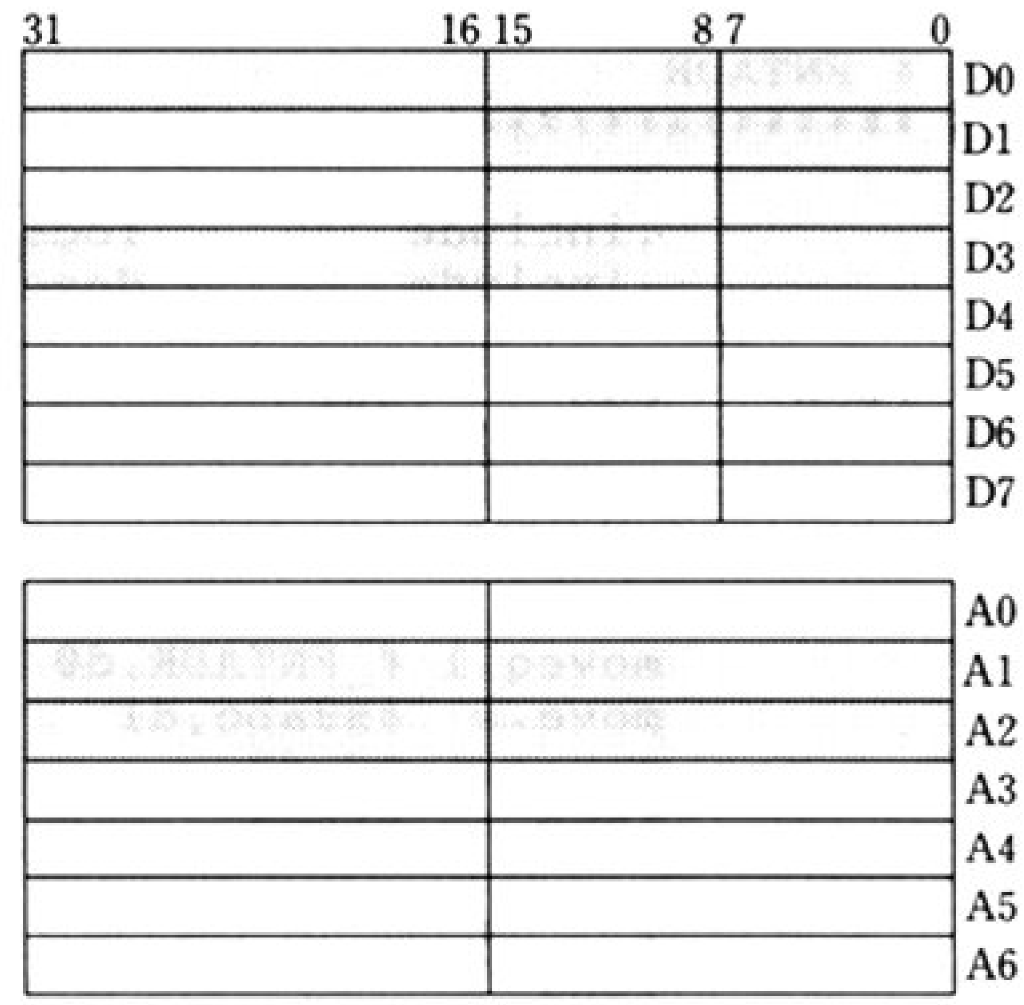
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_FNTADR,d0      * IOCSコール番号をセット
        move.w  #$9abc,d1        * シフトJIS コード
        moveq.l #8,d2            * 16×16 ドット
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

VRAMGET

● VRAMGET　　IOCS コール番号：$17

機　能　VRAM からバッファへのバイト単位読み込み



入　力

D0. L	$17	IOCS コール番号
A1. L	バッファの先頭アドレス	
A2. L	VRAM の先頭アドレス	
D1. W	(バッファの X 方向のバイト数)—1	
D2. W	(バッファの Y 方向のドット数)—1	
D3. L	(VRAM の X 方向のバイト数)—D1—1	

リターン　なし（D0、D1、D2、A1、A2 は内容が保証されません）

解　説　VRAM からバッファへ、バイト単位で読み込みます。

D1+D3 は（VRAM の X 方向のバイト数）—1 となります。

スーパーバイザ領域のアドレスを指定しますので、不正なアドレスを指定しないように注意してください。

VRAM にテキスト画面を指定する場合は、VRAM の X 方向バイト数は 128 バイトです。

プログラム例

```
**********************************************
* VRAMGET                                $17
* VRAMPUT                                $18
**********************************************

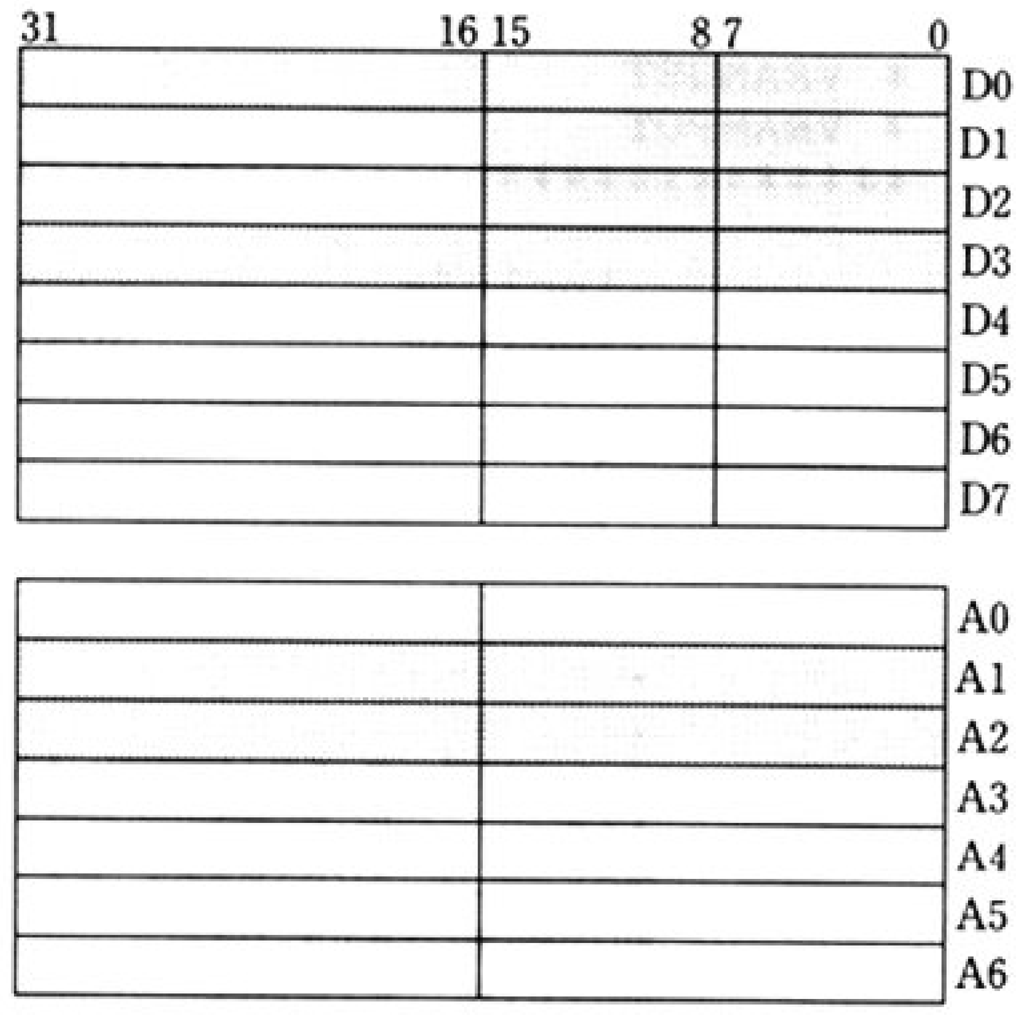
         .include        iocscall.mac
         .include        doscall.mac

iocs     equ      $0f
vramad1  equ      $e00000
vramad2  equ      $e00800
         .text
         .
         .
         moveq.l #_VRAMGET,d0      * IOCSコール番号をセット
         lea      buf,a1           * バッファの先頭アドレス
         movea.l #vramad1,a2       * VRAMのアドレス
         move.w   #15,d1           * バッファのX方向16byte
         move.w   #15,d2           * バッファのY方向16dot
         move.l   #112,d3          * VRAMのX方向のbyte数-D1-1
         trap     #iocs            * IOCSコール
         .
         .
         moveq.l #_VRAMPUT,d0      * IOCSコール番号をセット
         lea      buf,a1           * バッファの先頭アドレス
         movea.l #vramad2,a2       * VRAMのアドレス
         move.w   #15,d1           * バッファのX方向16byte
         move.w   #15,d2           * バッファのY方向16dot
         move.l   #112,d3          * VRAMのX方向のbyte数-D1-1
         trap     #iocs            * IOCSコール
         .
         .
         .data
         .
         .
buf:     ds.b     256
         .
         .
         .end
```

VRAMPUT

● VRAMPUT　　IOCS コール番号：$18

機　　能　バッファからVRAMへのバイト単位書き込み



入　　力

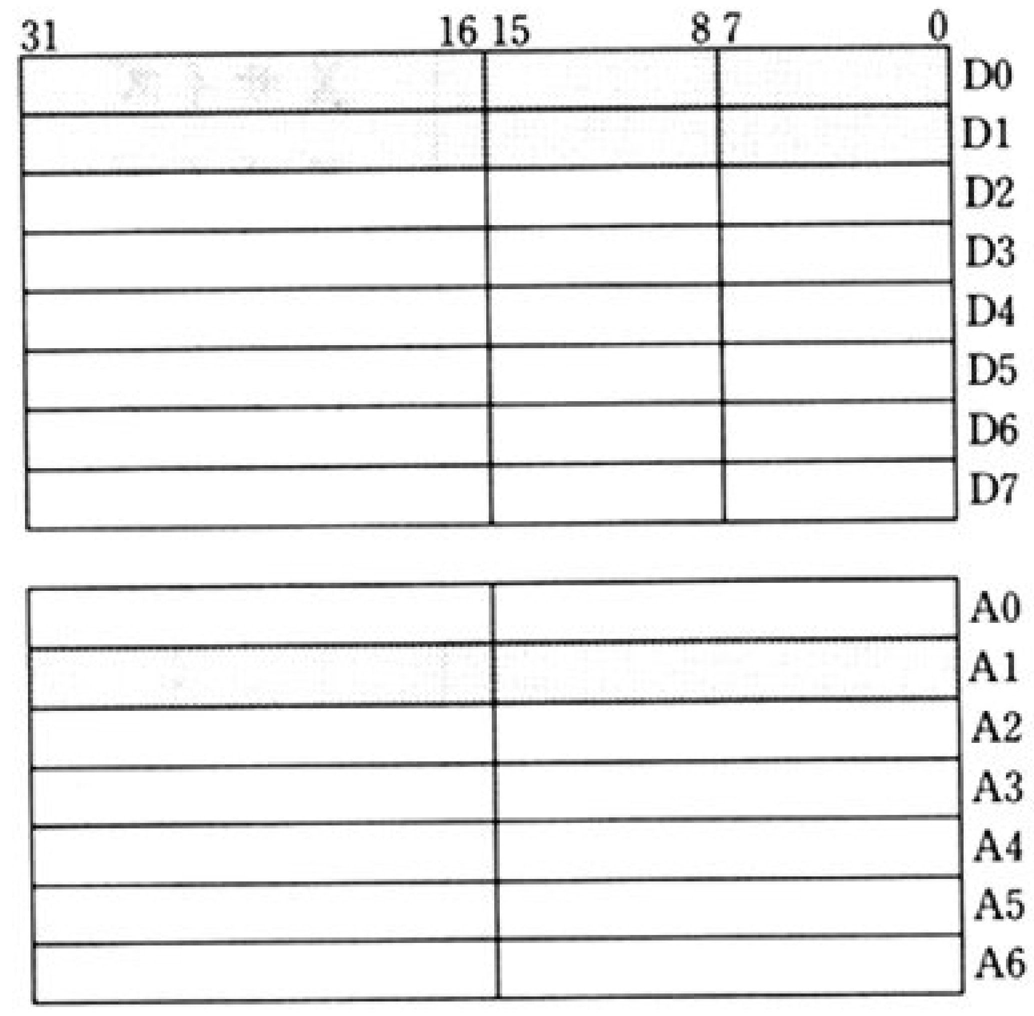
D0. L	$18	IOCS コール番号
A1. L	バッファの先頭アドレス	
A2. L	VRAMの先頭アドレス	
D1. W	(バッファのX方向のバイト数)−1	
D2. W	(バッファのY方向のドット数)−1	
D3. L	(VRAMのX方向のバイト数)−D1−1	

リターン　なし（D0、D1、D2、A1、A2は内容が保証されません）

解　　説　バッファからVRAMへ、バイト単位で書き込みます。
D1+D3は（VRAMのX方向のバイト数)−1となります。
スーパーバイザ領域のアドレスを指定しますので、不正なアドレスを指定しないように注意してください。
VRAMにテキスト画面を指定する場合は、VRAMのX方向のバイト数は128バイトです。
プログラム例はVRAMGET（$17）を参照してください。

FNTGET

● FNTGET　　IOCS コール番号：$19



機　　能　漢字パターンの指定アドレスへの読み込み

入　　力　D0. L　$19　IOCS コール番号
D1. L　漢字コード
A1. L　データバッファの先頭アドレス（4＋フォント分のバイト数が必要）

リターン　なし（D0. L は内容が保証されません）

0 (A1). W ＝ 文字パターンの X 方向のドット数
2 (A1). W ＝ 文字パターンの Y 方向のドット数
4 (A1). B ＝ パターンデータ

解　　説　指定の漢字パターンを、指定アドレスへ読み込みます。
入力の D1. L の上位 16 ビットで文字の大きさを、下位 16 ビットで漢字パターンの漢字コード（シフト JIS コード（JIS 漢字コード($2121～$7E7E)）も可能）を指定します。
文字の大きさの内容については次の通りです。

6…12×12、6×12 ドット
0, 8…16×16、8×16 ドット
12…24×24、12×24 ドット

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
各フォントパターンは、次の通り。

● 6×12

ビットD07　　　　　　D01　D00

Xサイズ　(上位)		+0
Xサイズ　(下位)		+1
Yサイズ　(上位)		+2
Yサイズ　(下位)		+3
フォントデータ	0　0	+4
:	0　0	+5
:	:	:
:	:	:
:	0　0	+14
:	0　0	+15
		バイト

● 8×16

ビットD07　　　　　　D01　D00

Xサイズ　(上位)	+0
Xサイズ　(下位)	+1
Yサイズ　(上位)	+2
Yサイズ　(下位)	+3
フォントデータ	+4
:	+5
:	:
:	:
:	+14
:	+15
	バイト

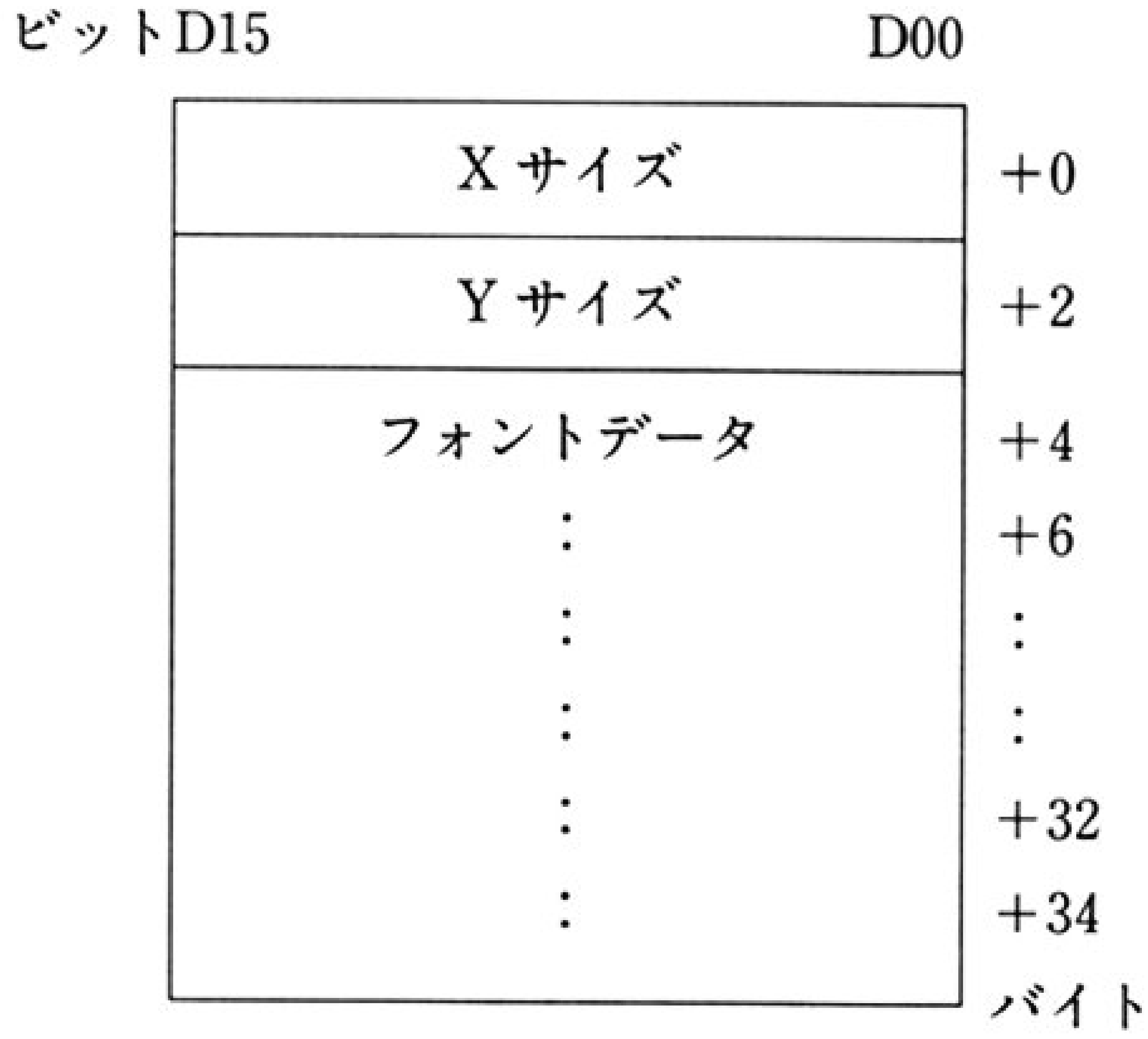
● 12×12

ビット D15	D03　　D00	
Xサイズ		+0
Yサイズ		+2
フォント	0 0 0 0	+4
データ	0 0 0 0	+6
:	:	:
:	:	:
:	0 0 0 0	+22
:	0 0 0 0	+24
		バイト

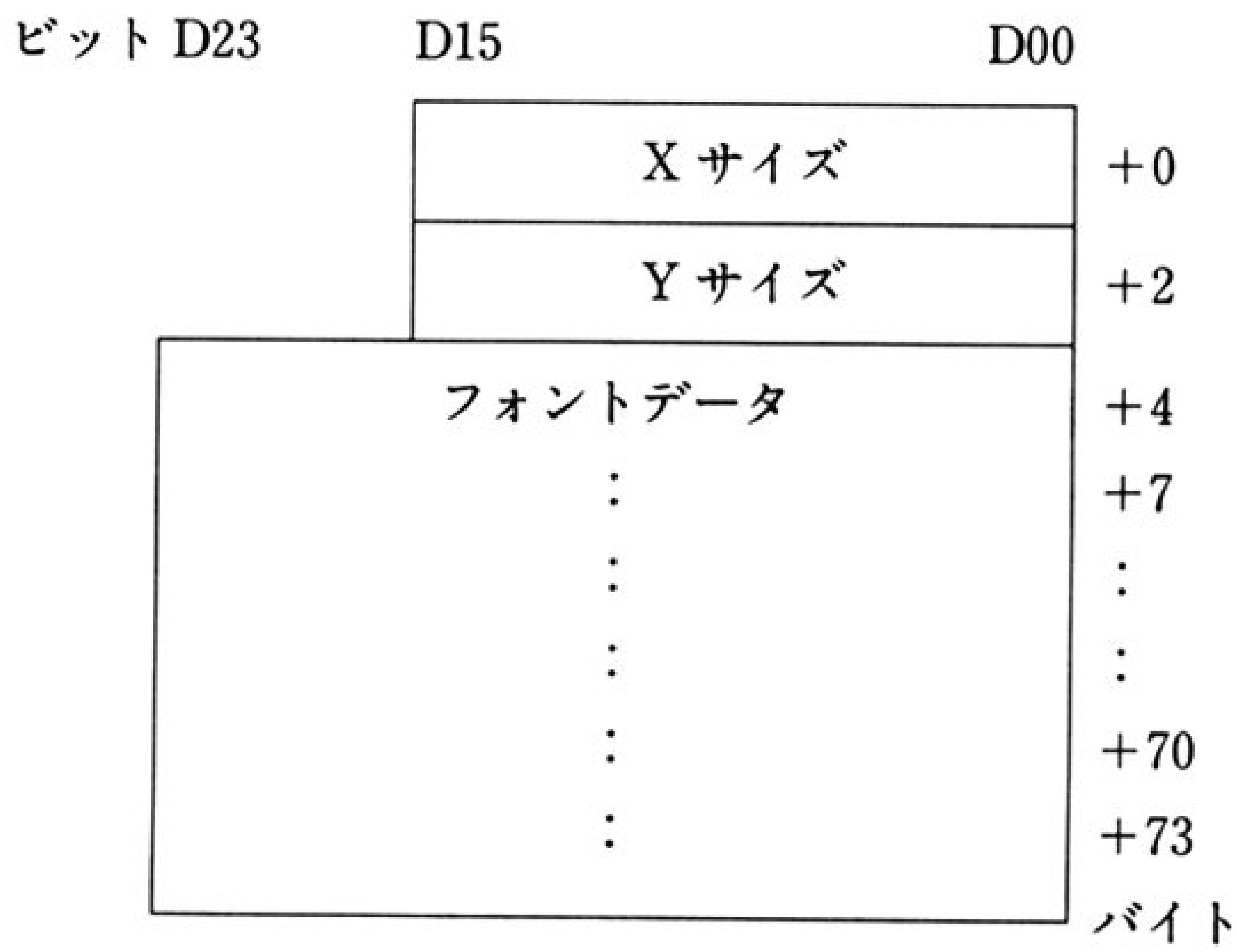
● 12×24

ビット D15	D03　　D00	
Xサイズ　(上位)		+0
Yサイズ　(上位)		+2
フォント	0 0 0 0	+4
データ	0 0 0 0	+6
:	:	:
:	:	:
:	0 0 0 0	+48
:	0 0 0 0	+50
		バイト

● 16×16


ビットD15
D00
Xサイズ
+0
Yサイズ
+2
フォントデータ
+4
+6
+32
+34
バイト

● 24×24


ビット D23
D15
D00
Xサイズ
+0
Yサイズ
+2
フォントデータ
+4
+7
+70
+73
バイト

プログラム例

```
*******************************************
* FNTGET                              $19
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        moveq.l #_TCOLOR,d0
        move.b  #%0001,d1           * テキストプレーン 0 指定
        trap    #iocs

        moveq.l #_FNTGET,d0         * IOCSコール番号をセット
        move.l  #$08_3444,d1        * フォントサイズ $08、漢字コード '環'
        lea     w_fnt,a1            * バッファアドレス
        trap    #iocs               * IOCSコール

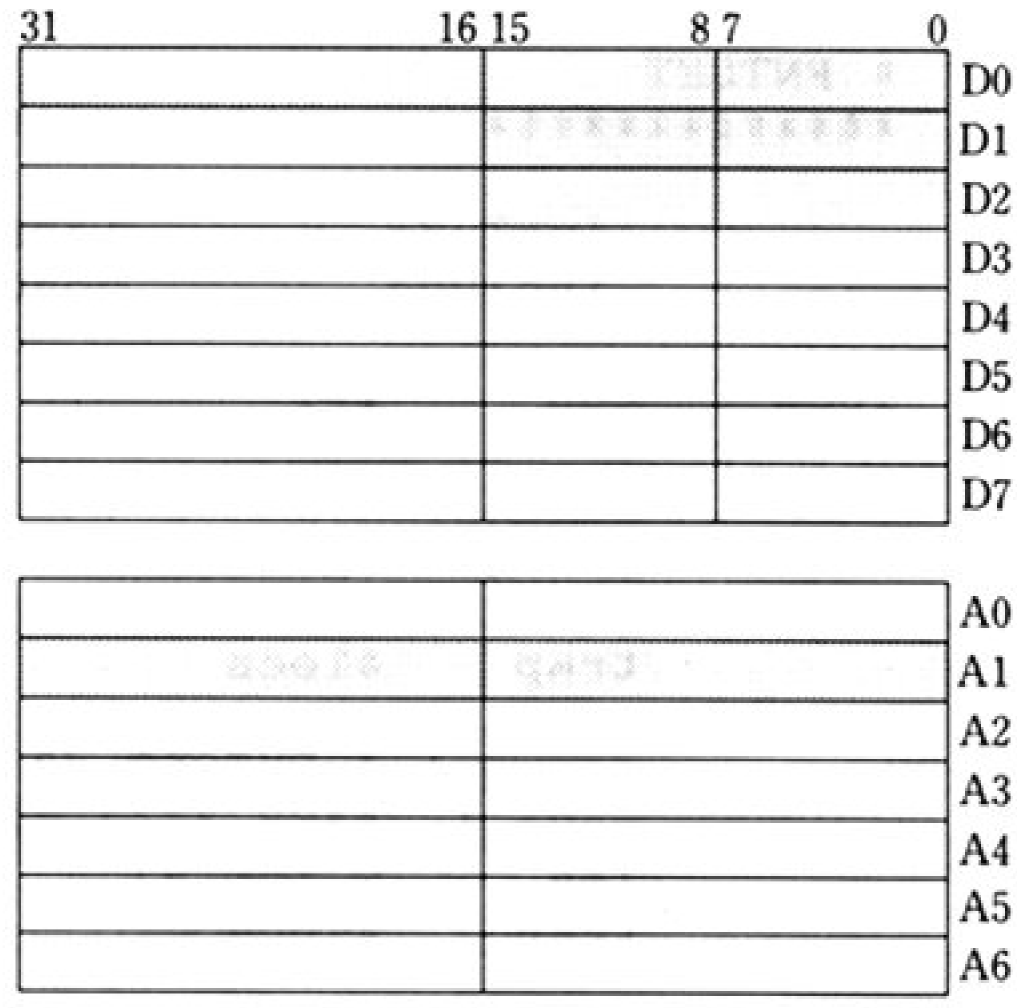
        moveq.l #_TEXTPUT,d0        * テキストに漢字を出力
        move.w  #100,d1             * X = 100 (dot)
        move.w  #200,d2             * Y = 200 (dot)
        lea     w_fnt,a1
        trap    #iocs

        dc.w    _EXIT

        .data
w_fnt:  ds.b    36                  * 4 + 16 * 16 / 8
        .end
```

TEXTGET

● TEXTGET　　IOCS コール番号：$1A



機　　能　ドット座標から指定アドレスへのパターン読み込み

入　　力

D0. L	$1A	IOCS コール番号
D1. W		X ドット座標
D2. W		Y ドット座標
A1. L		データバッファの先頭アドレス
0 (A1). W		読み込むパターンのX方向のドット数
2 (A1). W		読み込むパターンのY方向のドット数
4 (A1). B		パターンデータ用のバッファ

リターン　なし（D0. L は内容が保証されません）

解　　説　指定のドット座標から指定アドレスへ、パターンを読み込みます。

入力の D1. W、D2. W はそれぞれ読み込むパターンの X ドット座標、Y ドット座標です。

入力の A1. L は、データバッファの先頭アドレスです。

入力の 0 (A1). W、2 (A1). W はそれぞれ読み込むパターンの X 方向のドット数、Y 方向のドット数です。

入力の 4 (A1). B は、パターンデータ用のバッファです。

スーパーバイザ領域のアドレスも指定できますが、不正なアドレスを指定しないように注意してください。

プログラム例

```
********************************************
* TEXTGET                                $1A
********************************************

        .include        iocscall.mac
        .include        doscall.mac

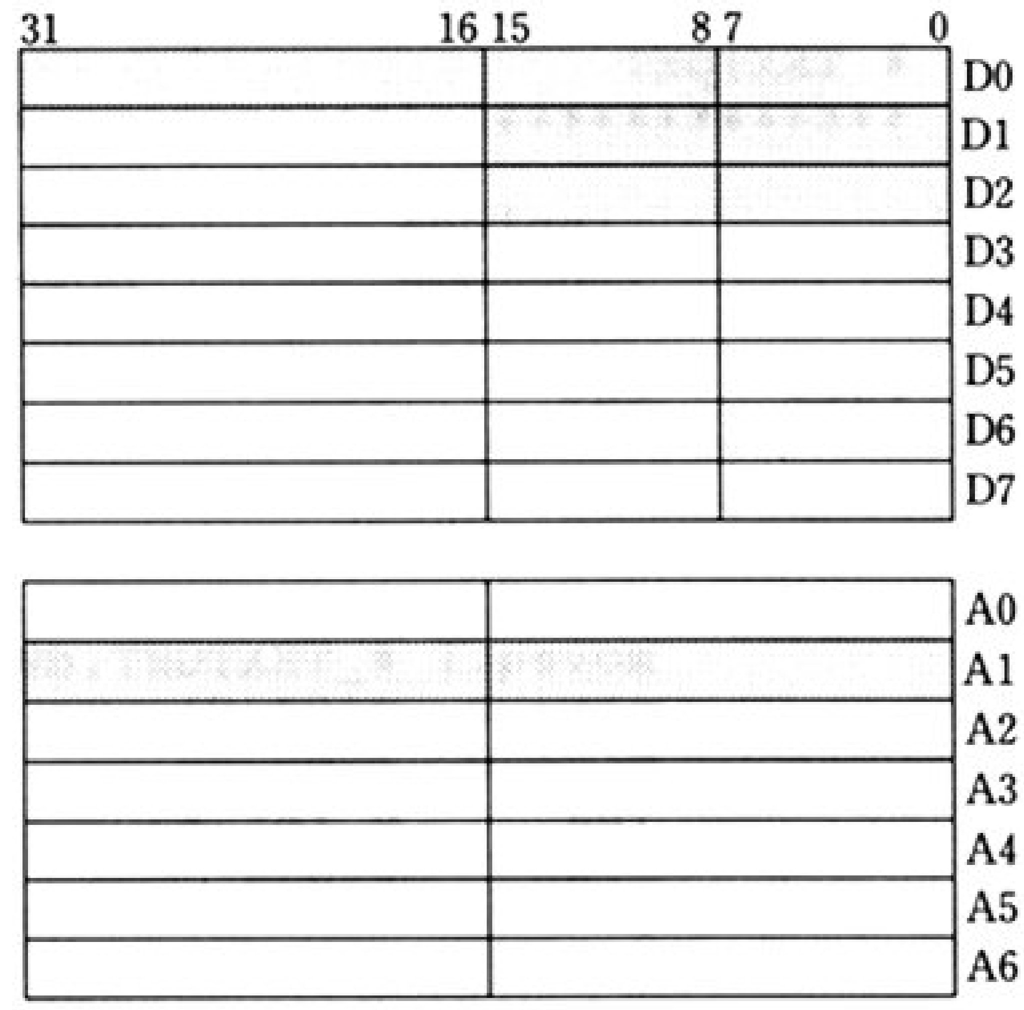
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TEXTGET,d0    * IOCSコール番号をセット
        move.w  #100,d1         * X = 100 (dot)
        move.w  #200,d2         * Y = 200 (dot)
        lea     w_fnt,a1        * バッファアドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
w_fnt:
        dc.w    16,16
        ds.w    32
        .
        .
        .end
```

TEXTPUT

● TEXTPUT　　IOCS コール番号：$1B

機　能　指定アドレスからドット座標へのパターン書き込み



入　力

D0. L	$1B	IOCS コール番号
D1. W		X ドット座標
D2. W		Y ドット座標
A1. L		データバッファの先頭アドレス
0 (A1). W		書き出すパターンのX方向のドット数
2 (A1). W		書き出すパターンのY方向のドット数
4 (A1). B		パターンデータ

リターン　なし（D0. L は内容が保証されません）

解　説　指定アドレスのパターンを、指定ドット座標に書き込みます。

入力の D1. W、D2. W は、それぞれ指定するパターンの X ドット座標、Y ドット座標です。

入力の A1. L は、データバッファの先頭アドレスです。

入力の 0 (A1). W、2 (A1). W は、それぞれ書き出すパターンの X 方向のドット数、Y 方向のドット数です。

入力の 4 (A1). B は、パターンデータです。

スーパーバイザ領域のアドレスも指定できますが、不正なアドレスを指定しないように注意してください。

プログラム例

```
*******************************************
* TEXTPUT                                    $1B
*******************************************

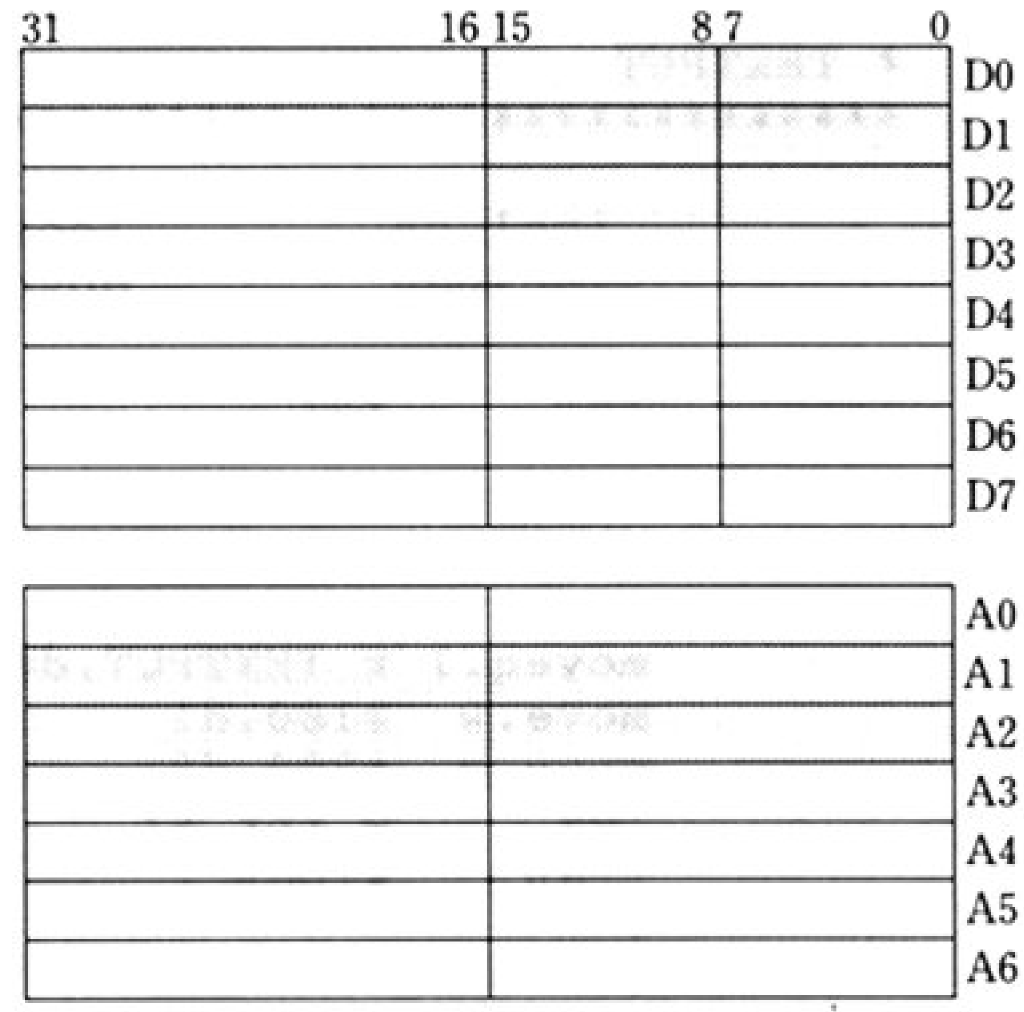
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TEXTPUT,d0    * IOCSコール番号をセット
        move.w  #100,d1         * X = 100 (dot)
        move.w  #200,d2         * Y = 200 (dot)
        lea     w_fnt,a1        * バッファアドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
w_fnt:
        dc.w    16,16
        dc.w    %1111111111111111
        dc.w    %0111111111111110
        dc.w    %0011111111111100
        dc.w    %0001111111111000
        dc.w    %0000111111110000
        dc.w    %0000011111100000
        dc.w    %0000001111000000
        dc.w    %0000000110000000
        dc.w    %0000000110000000
        dc.w    %0000001111000000
        dc.w    %0000011111100000
        dc.w    %0000111111110000
        dc.w    %0001111111111000
        dc.w    %0011111111111100
        dc.w    %0111111111111110
        dc.w    %1111111111111111
        .
        .
        .end
```

CLIPPUT

● CLIPPUT　　IOCS コール番号：$1C



機　能　指定アドレスからドット座標へのパターン書き込み（クリッピング処理）

入　力

D0. L	$1C	IOCS コール番号
D1. W	X ドット座標	
D2. W	Y ドット座標	
A1. L	データバッファの先頭アドレス	
0 (A1). W	書き込むパターンのX方向のドット数	
2 (A1). W	書き込むパターンのY方向のドット数	
4 (A1). B	パターンデータ	
A2. L	クリップ座標バッファの先頭アドレス	
0 (A2). W	表示領域の X 座標左端	
2 (A2). W	表示領域の Y 座標上端	
4 (A2). W	表示領域の X 座標右端	
6 (A2). W	表示領域の Y 座標下端	

リターン　なし（D0. L は内容が保証されません）

解　説　指定アドレスのパターンを、指定ドット座標に書き込みます。
指定したクリップ座標にしたがいクリッピング処理します。
入力の D1. W、D2. W は、それぞれ X ドット座標、Y ドット座標です。
入力の A1. L は、データバッファの先頭アドレスです。
入力の 0 (A1). W は、それぞれ書き込むパターンの X 方向のドット数、Y 方向のドット数です。
入力の 4 (A1). B は、パターンデータです。
入力の A2. L は、クリップ座標バッファの先頭アドレスです。
表示領域は（X1、Y1：左上端）－（X2、Y2：右下端）です。
X1、Y1、X2、Y2 は、それぞれ入力の 0 (A2). W、2 (A2). W、4 (A2). W、6 (A2). W です。
スーパーバイザ領域のアドレスも指定できますが、不正なアドレスを指定しないように注意してください。

プログラム例

```
*********************************************
* CLIPPUT                                $1C
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

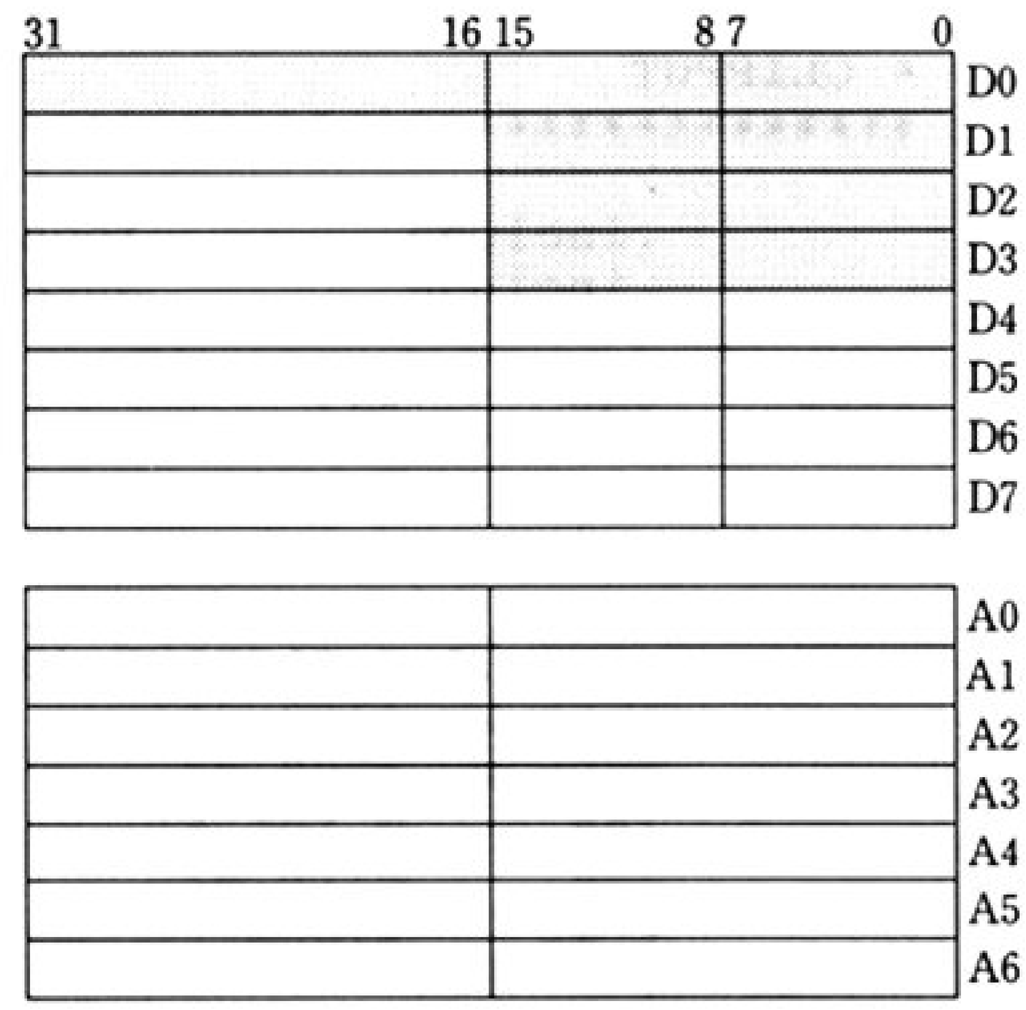
        .text
        .
        .
        moveq.l #_CLIPPUT,d0    * IOCSコール番号をセット
        move.w  #100,d1         * X = 100 (dot)
        move.w  #200,d2         * Y = 200 (dot)
        lea     w_fnt,a1        * バッファアドレス
        lea     clip,a2         * クリップ座標データアドレス
        trap    #iocs           * IOCSコール
        .
        .

clip:
        dc.w    50,50,250,250

w_fnt:  dc.w    95,95
        dc.b    ...........
        .
        .
        .
        .end
```

SCROLL

● SCROLL　　IOCS コール番号：$1D



機　能　テキスト／グラフィックの表示座標を設定／検査

入　力　D0. L　$1D　IOCS コール番号
D1. W　モード
D2. W　X ドット座標
D3. W　Y ドット座標

リターン　D0. L　座標

解　説　テキスト／グラフィックの表示座標を設定／検査します。
入力の D1. W はモードです。
内容は次の通りです。

D1. W	モード
0	グラフィックスクリーン 0 の設定
1	グラフィックスクリーン 1 の設定
2	グラフィックスクリーン 2 の設定
3	グラフィックスクリーン 3 の設定
4	グラフィックスクリーン 0 の検査
5	グラフィックスクリーン 1 の検査
6	グラフィックスクリーン 2 の検査
7	グラフィックスクリーン 3 の検査
8	テキストの設定
9	テキストの検査

入力の D2. W、D3. W は、それぞれ設定時の表示ドット座標の X、Y です。
リターンの D0. L の上位 16 ビット、下位 16 ビットはそれぞれ設定前／検査時のドット座標の X、Y です。

プログラム例

```
*******************************************
* SCROLL                              $1D
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

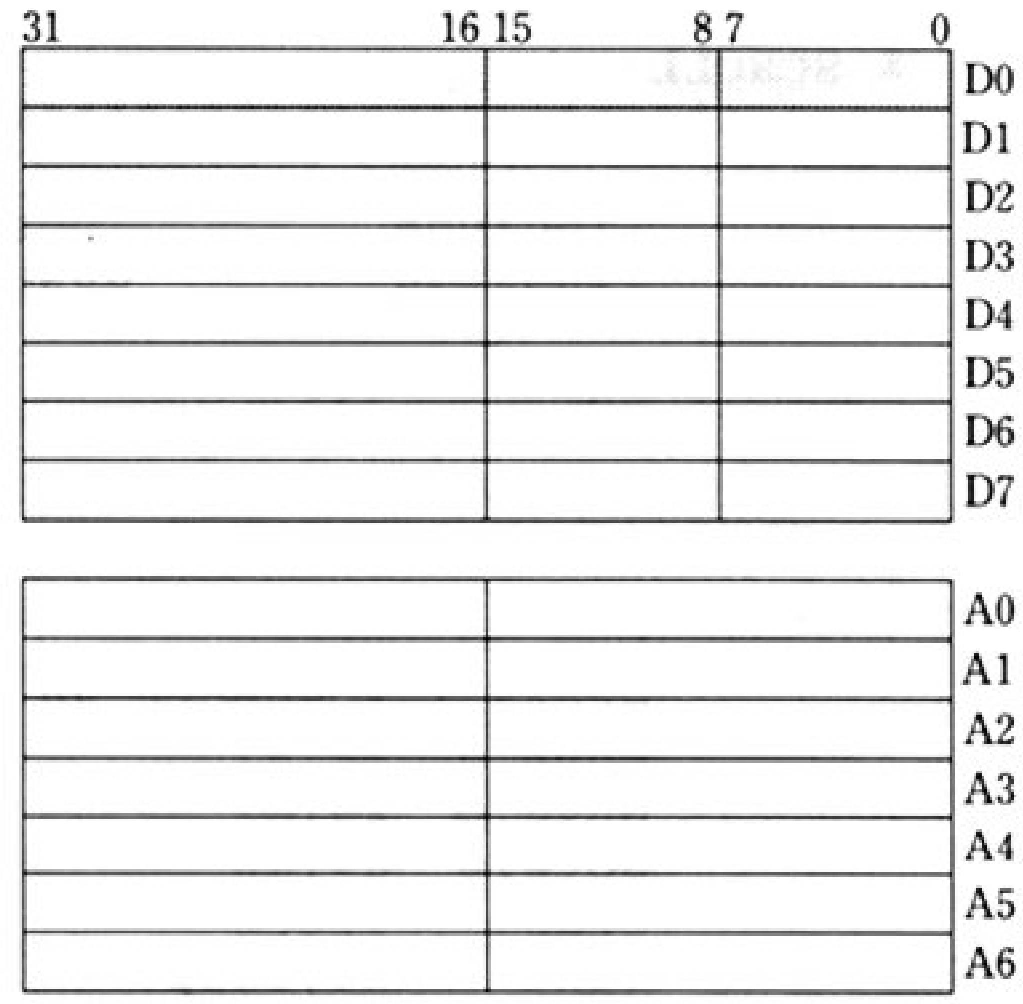
iocs    equ     $0f

        .text
        move.l  #1024,d4        * d4.l = 1024
loop:
        bsr     timer           * 速すぎるので、空ループを実行します
        moveq.l #_SCROLL,d0     * IOCSコール番号をセット
        move.w  #8,d1           * テキスト画面設定
        move.w  #0,d2           * X = 0
        move.w  d4,d3           * Y = d4.l
        trap    #iocs           * IOCSコール
        dbne    d4,loop         * --(d4.l) != 0 ならば loop へ
        dc.w    _EXIT

timer:
        move.w  #$2000,d0
tl:
        dbeq    d0,tl
        rts
        .end
```

B_CURON

● B_CURON　　IOCS コール番号：$1E



機　　能　カーソルを表示

入　　力　D0. L　$1E　IOCS コール番号

リターン　なし（D0. L は内容が保証されません）

解　　説　カーソルを表示します。

プログラム例

```
*******************************************
* B_CURON                             $1E
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_CURON,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_CUROFF

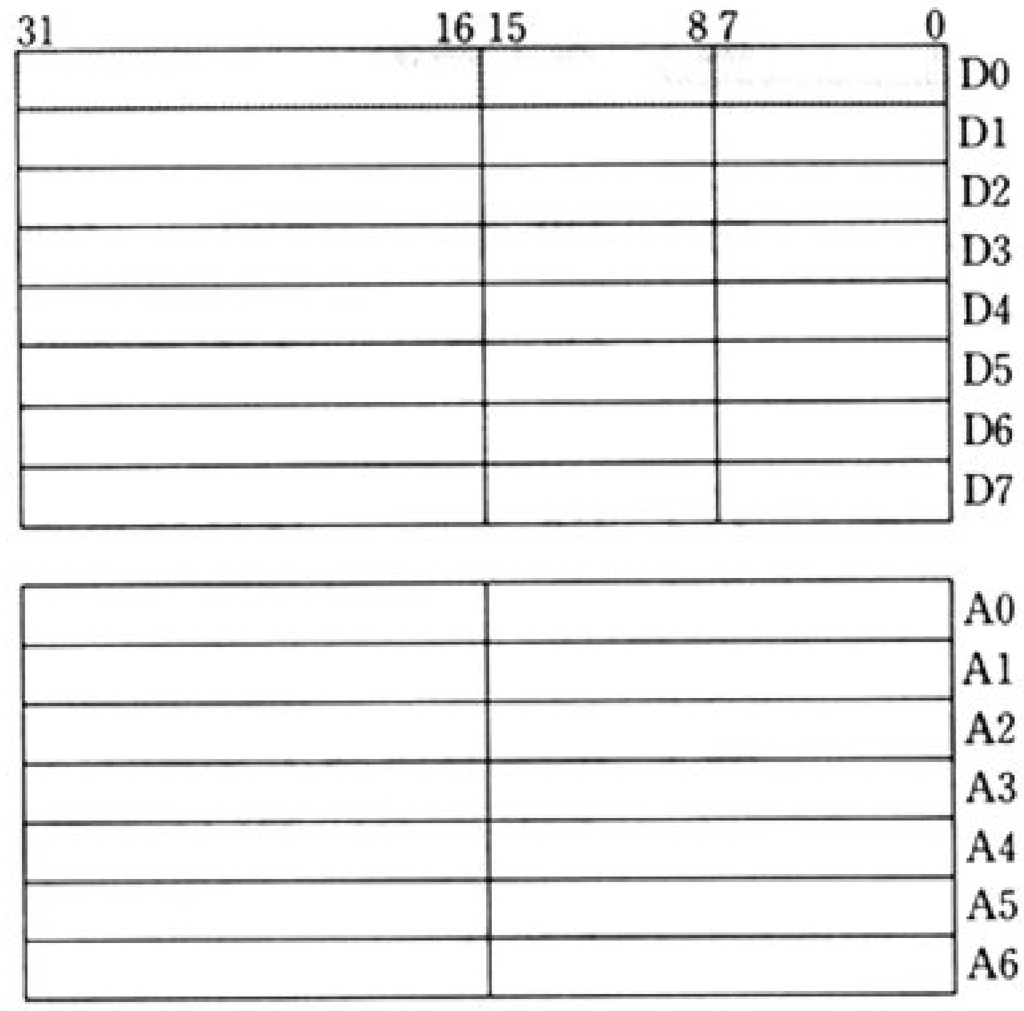
● B_CUROFF　　　IOCS コール番号：$1F

機　　能　カーソルを表示しない

入　　力　D0. L　　$1F　　IOCS コール番号

リターン　なし（D0. L は内容が保証されません）

解　　説　カーソルを表示しないようにします。



プログラム例

```
********************************************
* B_CUROFF                             $1F
********************************************

        .include        iocscall.mac
        .include        doscall.mac

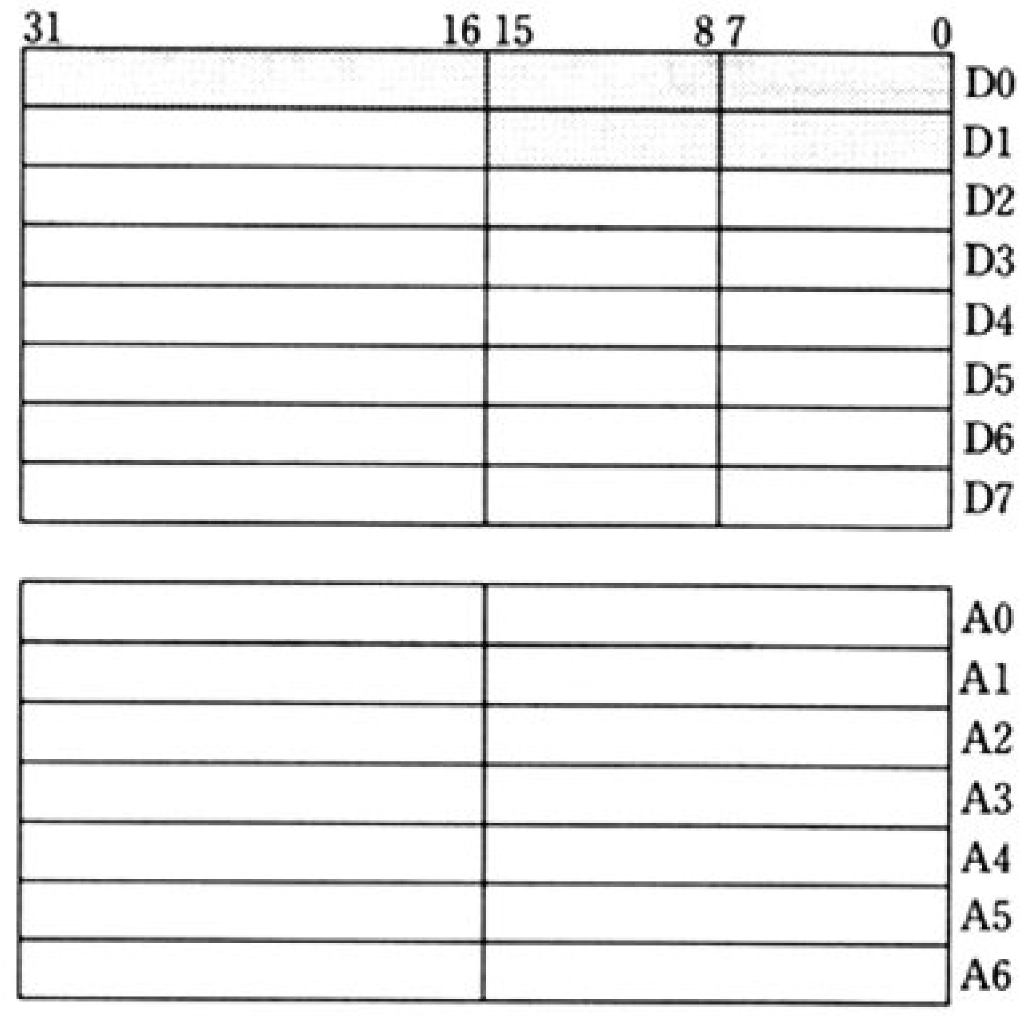
iocs    equ     $0f


        .text
        .
        .
        moveq.l #_B_CUROFF,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_PUTC

● B_PUTC　　IOCS コール番号：$20

機　　能　文字表示



入　　力　D0. L　$20　IOCS コール番号
D1. W　文字コード

リターン　D0. L　移動後のカーソル位置

解　　説　指定した文字を表示します。
入力の D1. W は、表示する文字のコードです。
文字が漢字の場合は第一バイト、第二バイトを二回に分けても、入力の D1. W に漢字コードをそのまま指定してもかまいません。
リターンの D0. L の上位 16 ビット、下位 16 ビットはそれぞれカーソル移動後のテキスト座標です。

プログラム例　D0. W にセットされたデータを出力するサブルーチンです (B_KEYINP ($00) 参照)。

```
*********************************************
* B_PUTC                                 $20
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

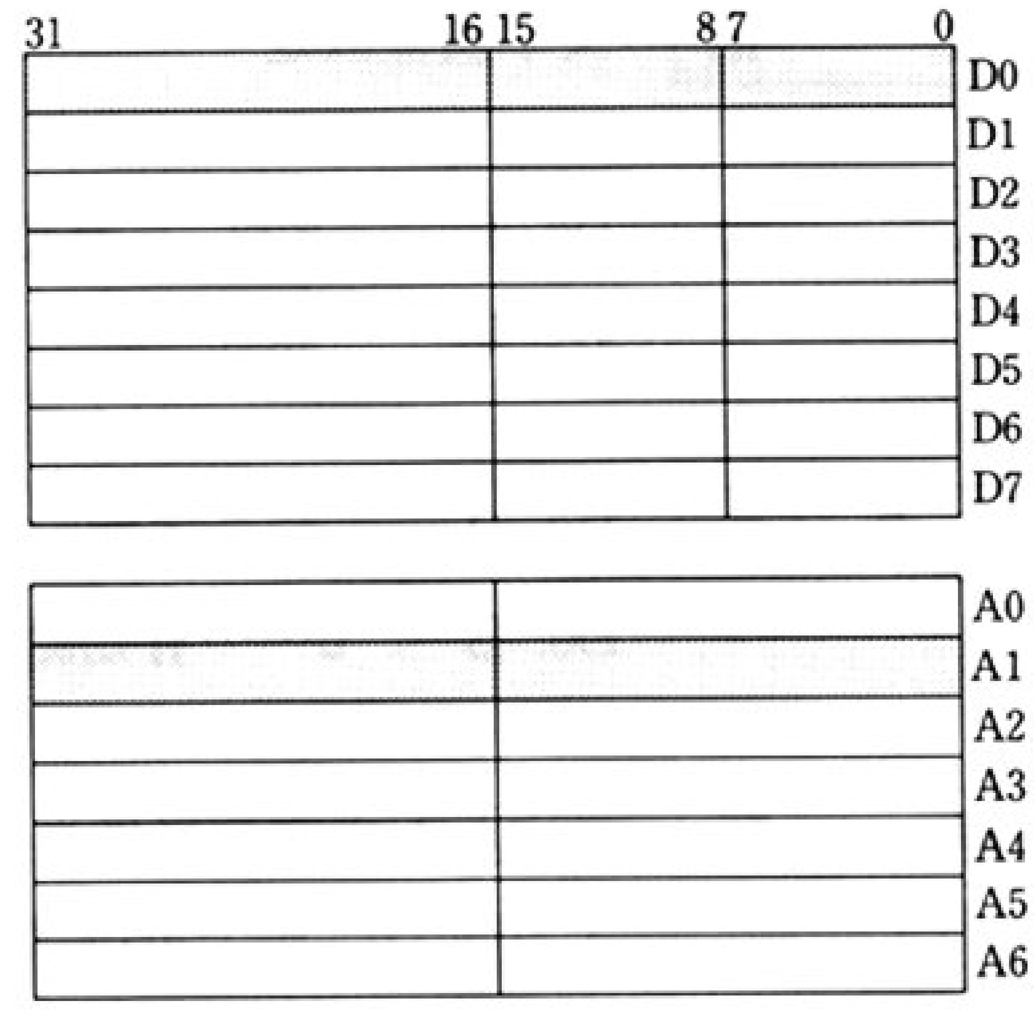
        .text
putchr:
        move.w  sr,-(sp)            * ステータスレジスタ退避
        movem.l d0-d1,-(sp)         * d0, d1 退避

        move.w  d0,d1               * 表示文字セット
        moveq.l #_B_PUTC,d0         * IOCSコール番号をセット
        trap    #iocs               * IOCSコール

        movem.l (sp)+,d0-d1         * d0 復帰
        rtr                         * 呼び出し元に帰ります
        .end
```

B_PRINT

● B_PRINT　　　IOCS コール番号：$21



機　　能　文字列表示

入　　力　D0. L　$21　IOCS コール番号
A1. L　文字列先頭アドレス

リターン　D0. L　移動後のカーソル位置
A1. L　文字列の終端（ヌル文字）の次のアドレス

解　　説　指定した文字列を表示します。
入力の A1. L は文字列の先頭アドレスです。
この文字列はヌル文字（0）で終わらなければなりません。
なお終端のヌル文字（0）は表示されません。
リターンの D0. L の上位 16 ビット下位 16 ビットは、それぞれカーソル移動後のテキスト座標です。

プログラム例

```
*******************************************
* B_PRINT                             $21
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

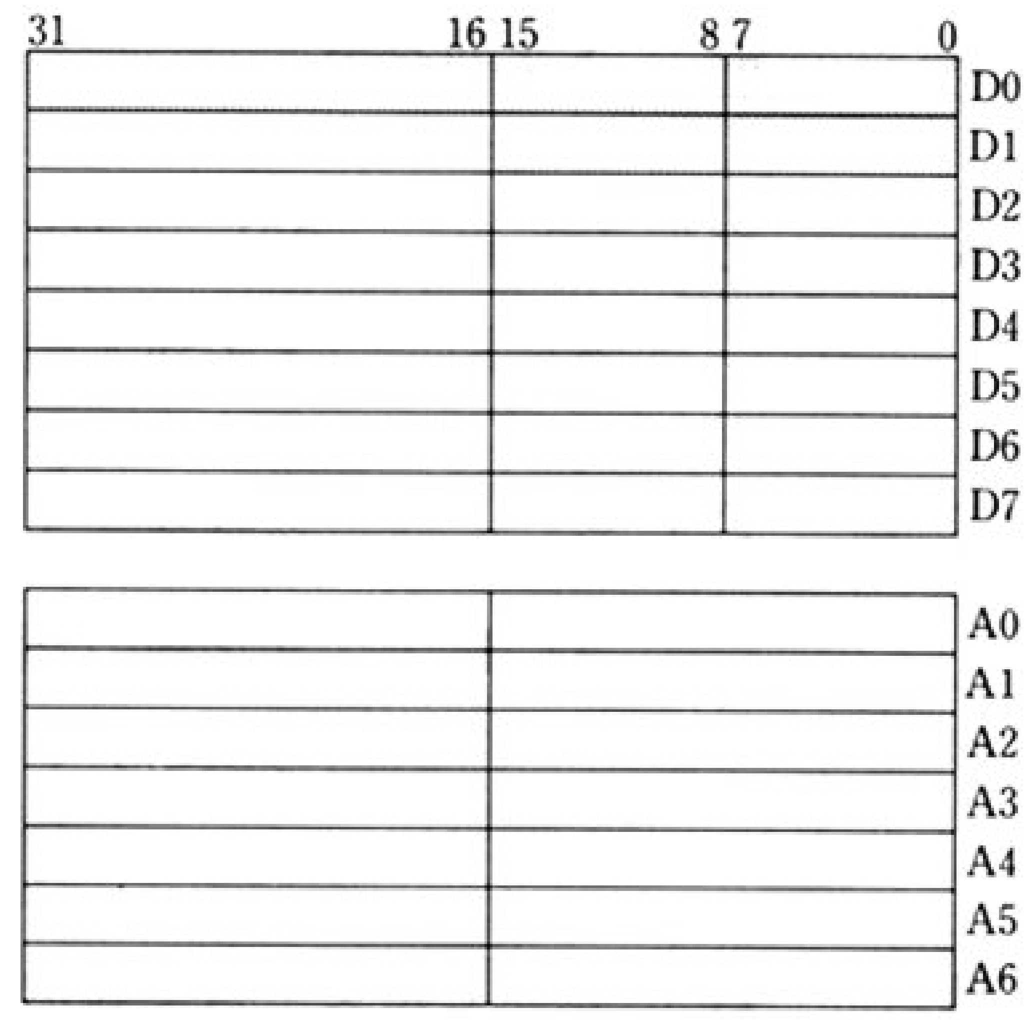
iocs    equ     $0f

        .text
        moveq.l #_B_PRINT,d0    * IOCSコール番号をセット
        lea     w_msg,a1        * a1.l に文字列の先頭アドレスをセット
        trap    #iocs           * IOCSコール
        dc.w    _EXIT

*---[ Data ]---*
        .data
w_msg:  dc.b    "Welcome to the X68000 World",$0d,$0a,0
        .end
```

B_COLOR

● B_COLOR　　IOCS コール番号：$22



機　　能　文字属性設定

入　　力　D0. L　$22　IOCS コール番号
D1. W　属性

リターン　D0. B　前の属性
D0. B ＜ 0……有効範囲外を指定した

解　　説　指定した文字の属性をセットします。
入力の D1. W は指定する属性です。
ここに−1 を指定すると D0. B に属性を返します。
指定できる属性の内容は次の通りです。

ビット3	ビット2	ビット1	ビット0	意　味
0	0	0	0	黒
0	0	0	1	水色
0	0	1	0	黄色
0	0	1	1	白
0	1	0	0	黒
0	1	0	1	水色・強調
0	1	1	0	黄色・強調
0	1	1	1	白・強調
1	0	0	0	黒
1	0	0	1	水色・リバース
1	0	1	0	黄色・リバース
1	0	1	1	白・リバース
1	1	0	0	黒
1	1	0	1	水色・強調・リバース
1	1	1	0	黄色・強調・リバース
1	1	1	1	白・強調・リバース

リターンの D0. B は前の属性です。
−1 のときはエラーです。
属性の内容は入力の D1. W と同じです。

プログラム例　文字の属性を変えてメッセージを出力します。
doscall. mac、iocscall. mac は、システムコールや IOCS コールの名前を定義したインクルードファイルです。

```
*******************************************
* B_COLOR                              $22
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        moveq.l #%1110,d0       * 黄色reverse HighLite
        bsr     color
        lea     msg,a1
        bsr     putstr
        moveq.l #%0011,d0       * 白色normal
        bsr     color
        lea     msg,a1
        bsr     putstr
        dc.w    _EXIT

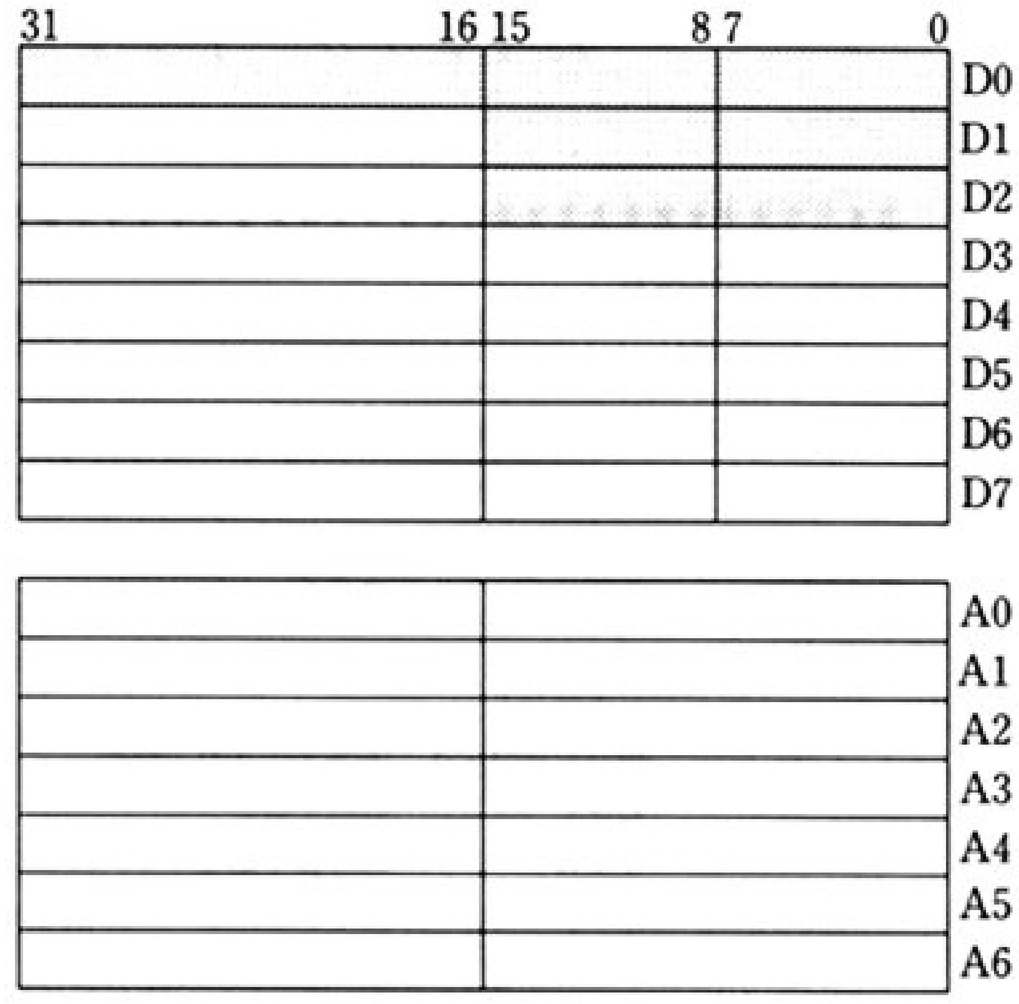
*-------------------------------------------
** COLOR
** in d0.w 属性
*       bit
*       0-1: Color (00:黒 01:水色 10:黄色 11:白)
*         2: HighLight
*         3: Reverse
*-------------------------------------------
color:
        movem.l d0-d1,-(sp)
        moveq.w d0,d1
        move.l  #_B_COLOR,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        movem.l (sp)+,d0-d1
        rts

putstr:
        move.l  d0,-(sp)
        moveq.l #_B_PRINT,d0
        trap    #iocs
        move.l  (sp)+,d0
        rts

        .data
msg:
        dc.b    "Hello world",$0d,$0a,0
        .end
```

B_LOCATE

● B_LOCATE　　IOCS コール番号：$23



機　能	カーソル移動		
入　力	D0. L	$23	IOCS コール番号
	D1. W	桁位置	
	D2. W	行位置	
リターン	D0. L	移動前のカーソル位置	

解　説

カーソルを指定した位置に移動します。
入力の D1. W と D2. W は、それぞれカーソルの移動先の桁と行位置です。
入力の D1. W に−1 を指定すると、リターンの D0. L に座標を返すだけで移動はしません。
リターンの D0. L の上位 16 ビットと下位 16 ビットは、それぞれカーソル移動前の桁と行位置です。

プログラム例 カーソルを目的位置に移動するサブルーチンです。
サブルーチンの独立性を高くするために、引数をスタックに積んで渡しています。

```
*******************************************
* B_LOCATE                             $23
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

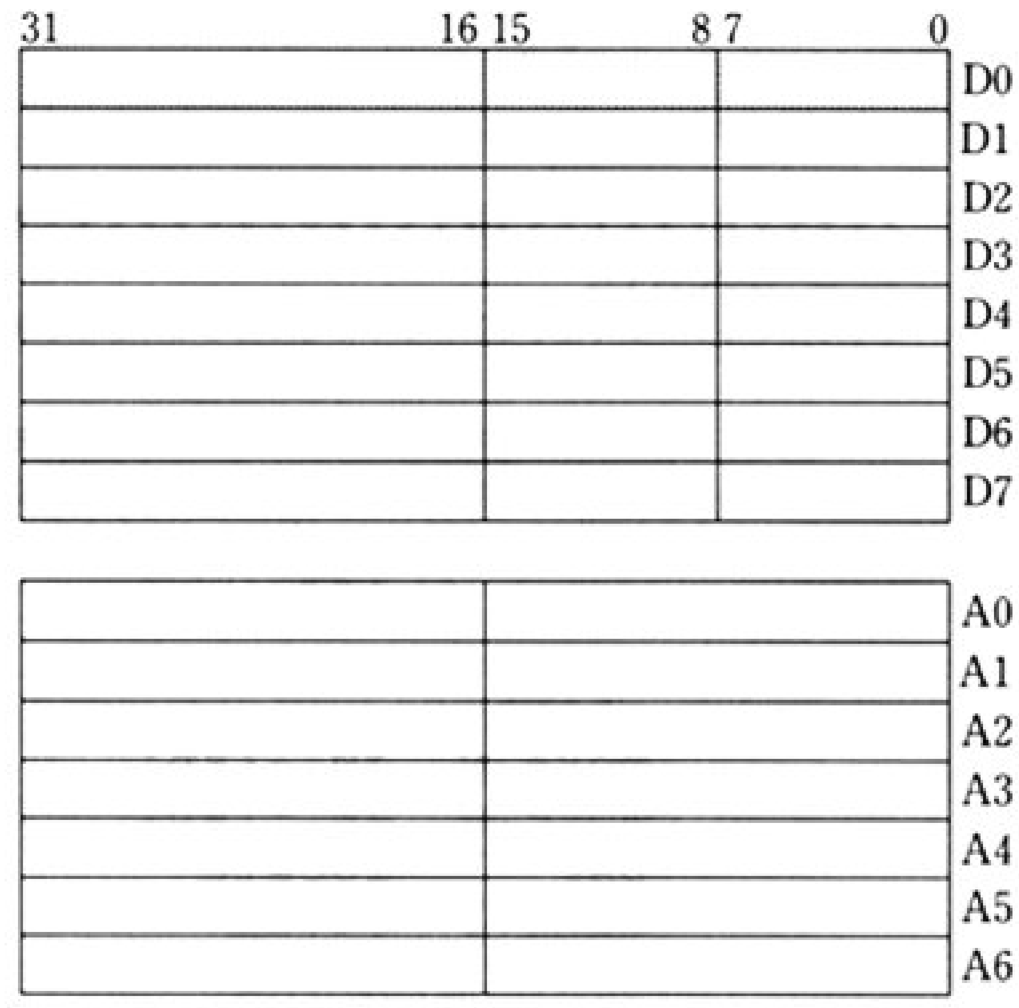
iocs    equ     $0f

        .text
        move.w  #0,-(sp)        * 行番号をプッシュ
        move.w  #0,-(sp)        * 桁をプッシュ
        bsr     locate          * サブルーチンコール
        addq.l  #4,sp           * スタックポインタを調節
        dc.w    _EXIT           * プロセス終了

locate:
        move.w  sr,-(sp)
        link    a6,#0
        movem.l d0-d2,-(sp)     * レジスタの退避
        moveq.l #_B_LOCATE,d0   * IOCSコール番号をセット
        move.w  10(a6),d1       * 桁
        move.w  12(a6),d2       * 行
        trap    #iocs           * IOCSコール
        movem.l (sp)+,d0-d2     * レジスタの復帰
        unlk    a6
        rtr
        .end
```

B_DOWN_S

● B_DOWN_S　　　IOCS コール番号：$24



機　能	カーソルを1行下へ移動
入　力	D0. L　$24　　IOCS コール番号
リターン	なし（D0. Lの内容は保証されません）
解　説	カーソルを1行下へ移動します。 カーソルが最下行にある場合はスクロールアップします。
プログラム例	カーソルを1行下に移動するサブルーチンです。 ステータスレジスタ(SR)を含むすべてのレジスタが保存されます。

```
*********************************************
* B_DOWN_S                               $24
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

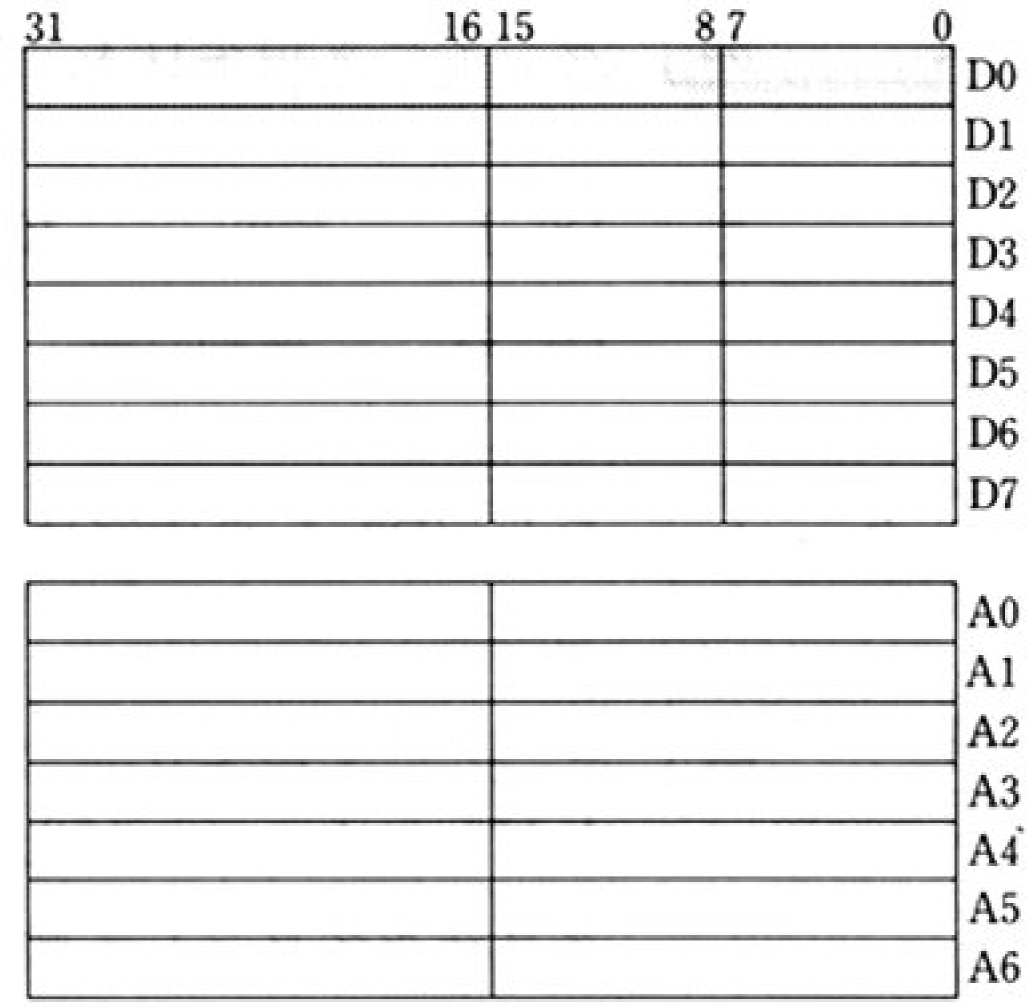
        .text
bdowns:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        move.l  d0,-(sp)        * d0 退避

        moveq.l #_B_DOWN_S,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール

        move.l  (sp)+,d0        * d0 復帰
        rtr                     * 呼び出し元に帰ります
        .end
```

B_UP_S

● B_UP_S　　IOCS コール番号：$25



機　　能　カーソルを1行上へ移動

入　　力　D0. L　$25　IOCS コール番号

リターン　なし（D0. L の内容は保証されません）

解　　説　カーソルを1行上へ移動します。
カーソルが先頭行にある場合はスクロールダウンします。

プログラム例　カーソルを1行上に移動するサブルーチンです。
ステータスレジスタ（SR）を含むすべてのレジスタが保存されます。

```
*******************************************
* B_UP_S                              $25
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
bups:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        move.l  d0,-(sp)        * d0 退避

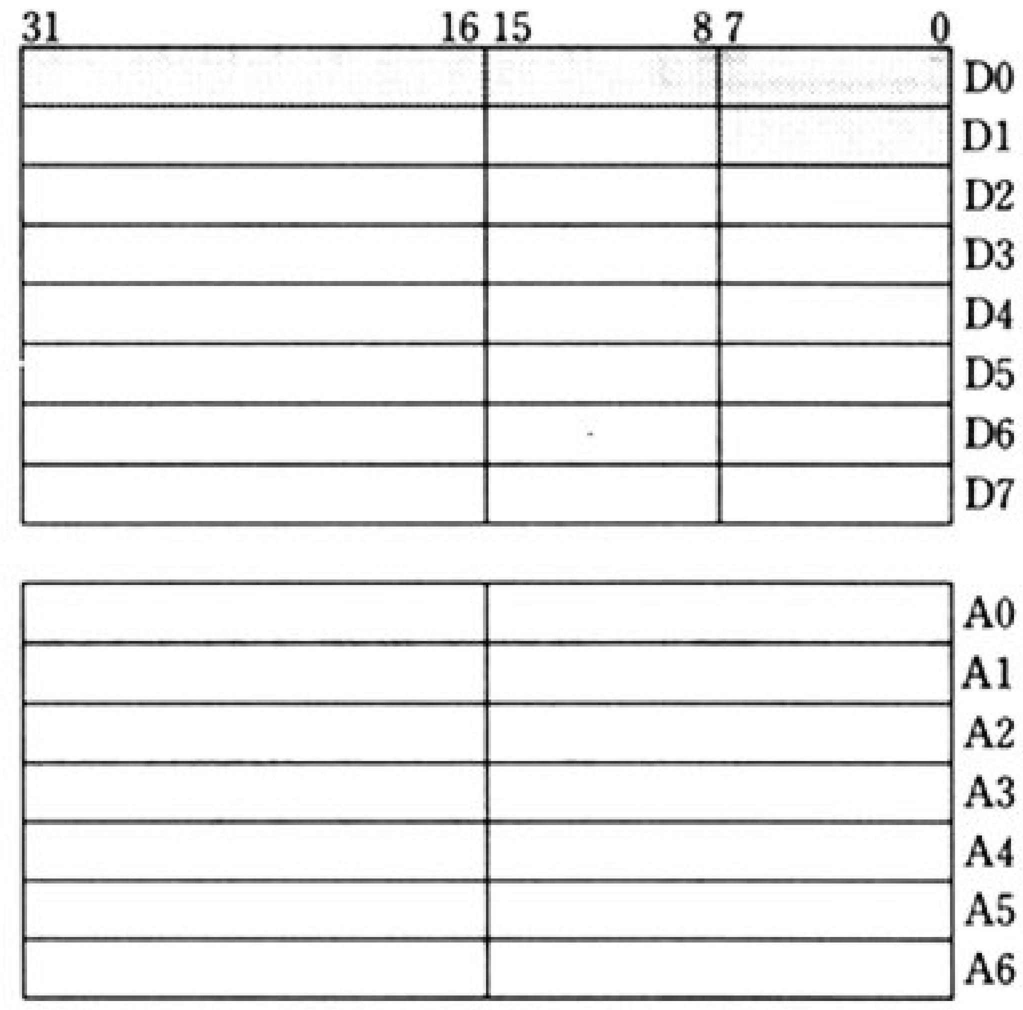
        moveq.l #_B_UP_S,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール

        move.l  (sp)+,d0        * d0 復帰
        rtr                     * 呼び出し元に帰ります
        .end
```

B_UP

● B_UP　　IOCS コール番号：$26

機能　カーソルを指定行上へ移動



入力　D0. L　$26　IOCS コール番号
D1. B　移動行数

リターン　なし（D0. L の内容は保証されません）

解説　カーソルを指定行上へ移動します。
ただし、カーソルが先頭行を越えてもスクロールダウンしません。
入力の D1. B は、移動行数を指定します。
0 を指定した場合は、1 と見なされます。

プログラム例　カーソルを指定行上に移動するサブルーチンです。
ステータスレジスタ（SR）を含むすべてのレジスタが保存されます。

```
********************************************
* B_UP                                  $26
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

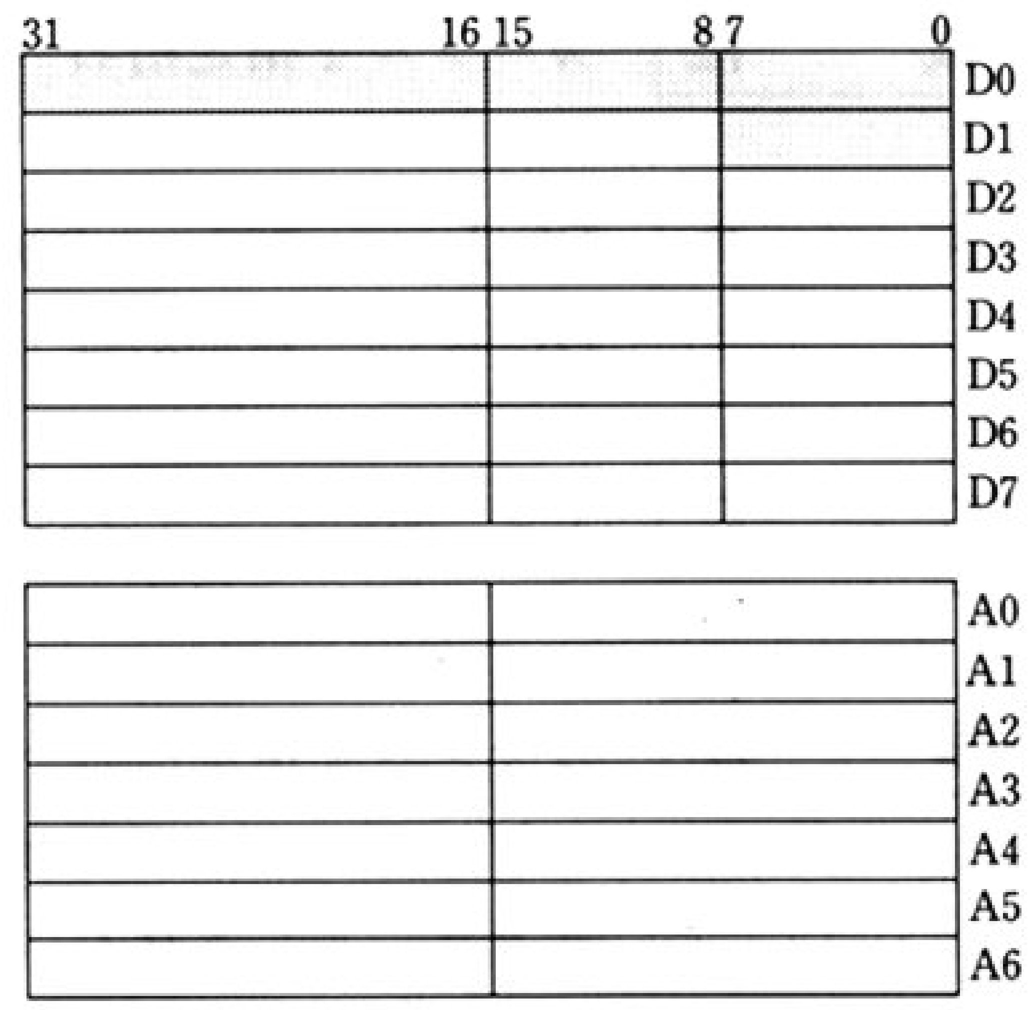
        .text
bup:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        link    a6,#0
        movem.l d0-d1,-(sp)     * d0、d1 退避

        moveq.l #_B_UP,d0       * IOCSコール番号をセット
        move.b  10(a6),d1       * 移動行数セット
        trap    #iocs           * IOCSコール

        movem.l (sp)+,d0-d1     * d0 復帰
        unlk    a6
        rtr                     * 呼び出し元に帰ります
        .end
```

B_DOWN

● B_DOWN　　　IOCS コール番号：$27



機　　能　カーソルを指定行下へ移動

入　　力　D0. L　　$27　　IOCS コール番号
D1. B　　移動行数

リターン　なし（D0. Lの内容は保証されません）

解　　説　カーソルを指定行下へ移動します。
ただし、カーソルが最下行を越えてもスクロールアップしません。
入力の D1. B は、移動行数を指定します。
0 を指定した場合は、1 と見なされます。

プログラム例　カーソルを指定行下に移動するサブルーチンです。
ステータスレジスタ（SR）を含むすべてのレジスタが保存されます。

```
********************************************
* B_DOWN                                 $27
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
bdown:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        link    a6,#0
        movem.l d0-d1,-(sp)     * d0、d1 退避

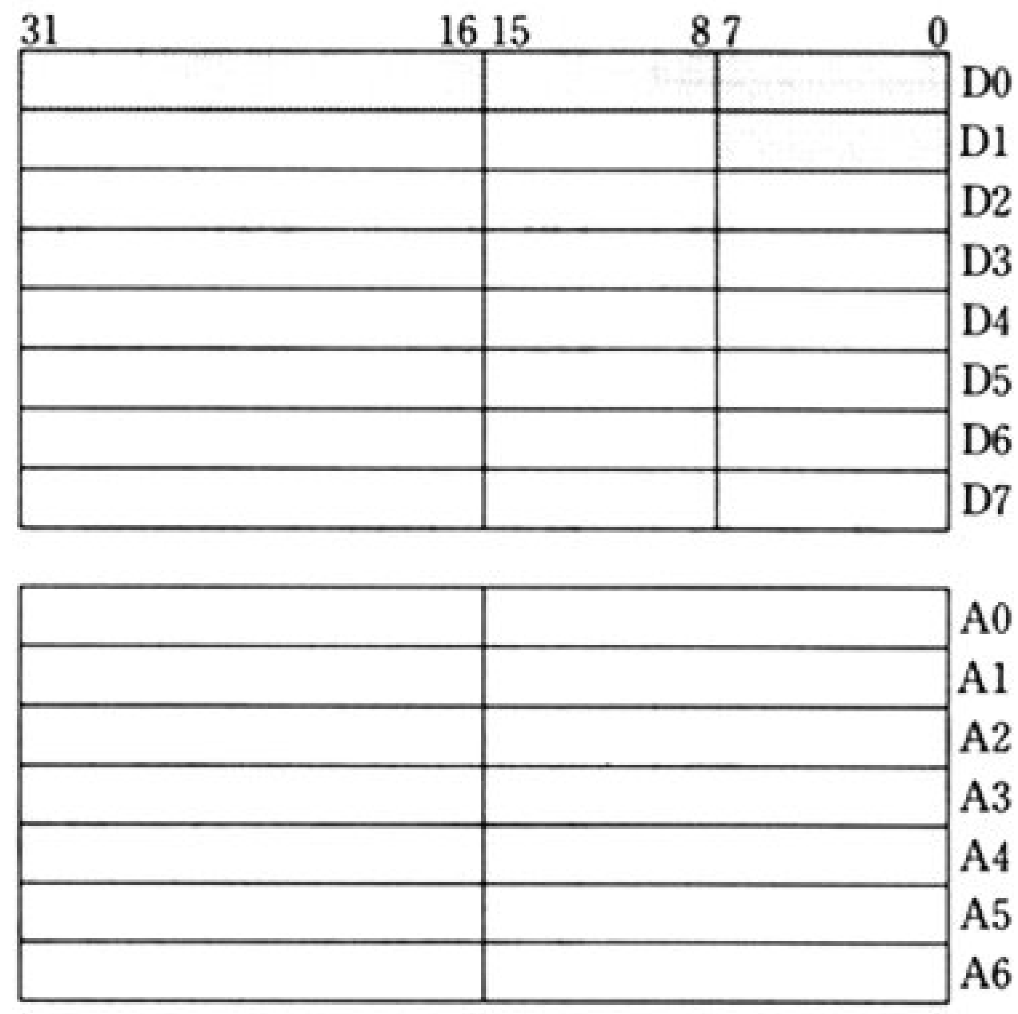
        moveq.l #_B_DOWN,d0     * IOCSコール番号をセット
        move.b  10(a6),d1       * 移動行数セット
        trap    #iocs           * IOCSコール

        movem.l (sp)+,d0-d1     * d0 復帰
        unlk    a6
        rtr                     * 呼び出し元に帰ります
        .end
```

B_RIGHT

● B_RIGHT　　IOCS コール番号：$28

機　能　カーソルを指定桁右へ移動



入　力　D0. L　$28　IOCS コール番号
D1. B　移動桁数

リターン　なし（D0. L の内容は保証されません）

解　説　カーソルを指定桁右へ移動します。
ただしカーソルが画面の右端を越えても
スクロールしません。
入力の D1. B は移動桁数を指定します。
0 を指定した場合は、1 と見なされます。

プログラム例　カーソルを指定桁右に移動するサブルーチンです。
ステータスレジスタ（SR）を含むすべてのレジスタが保存されます。

```
*****************************************
* B_RIGHT                            $28
*****************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

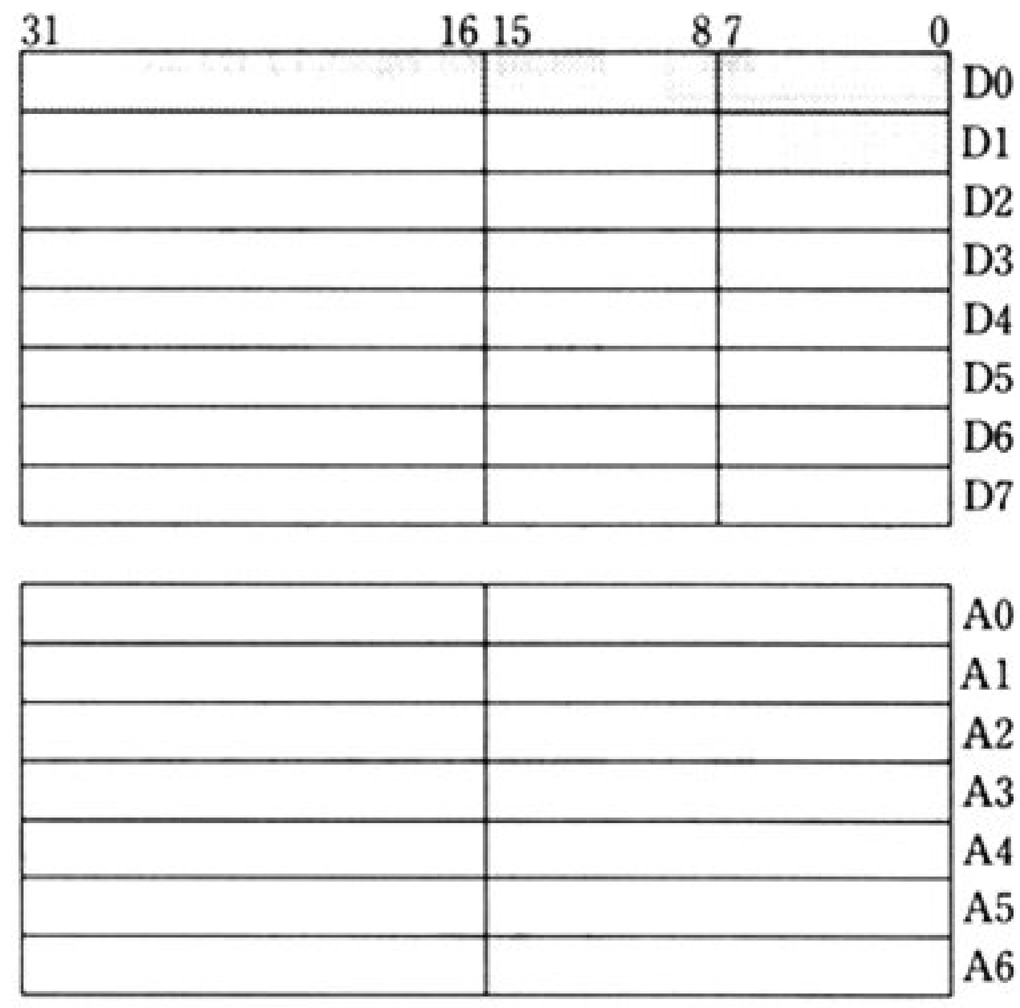
        .text
bright:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        link    a6,#0
        movem.l d0-d1,-(sp)     * d0、d1 退避

        moveq.l #_B_RIGHT,d0    * IOCSコール番号をセット
        move.b  10(a6),d1       * 移動桁数セット
        trap    #iocs           * IOCSコール

        movem.l (sp)+,d0-d1     * d0 復帰
        unlk    a6
        rtr                     * 呼び出し元に帰ります
        .end
```

B_LEFT

● B_LEFT　　　IOCS コール番号：$29



機　能　カーソルを指定桁左へ移動

入　力　D0. L　$29　IOCS コール番号
D1. B　移動桁数

リターン　なし（D0. L の内容は保証されません）

解　説　カーソルを指定桁左へ移動します。
ただし、カーソルが画面の左端を越えてもスクロールしません。
入力の D1. B は、移動桁数を指定します。
0 を指定した場合は、1 と見なされます。

プログラム例　カーソルを指定桁左に移動するサブルーチンです。
ステータスレジスタ（SR）を含むすべてのレジスタが保存されます。

```
*********************************************
* B_LEFT                                 $29
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
bleft:
        move.w  sr,-(sp)        * ステータスレジスタ退避
        link    a6,#0
        movem.l d0-d1,-(sp)     * d0、d1 退避

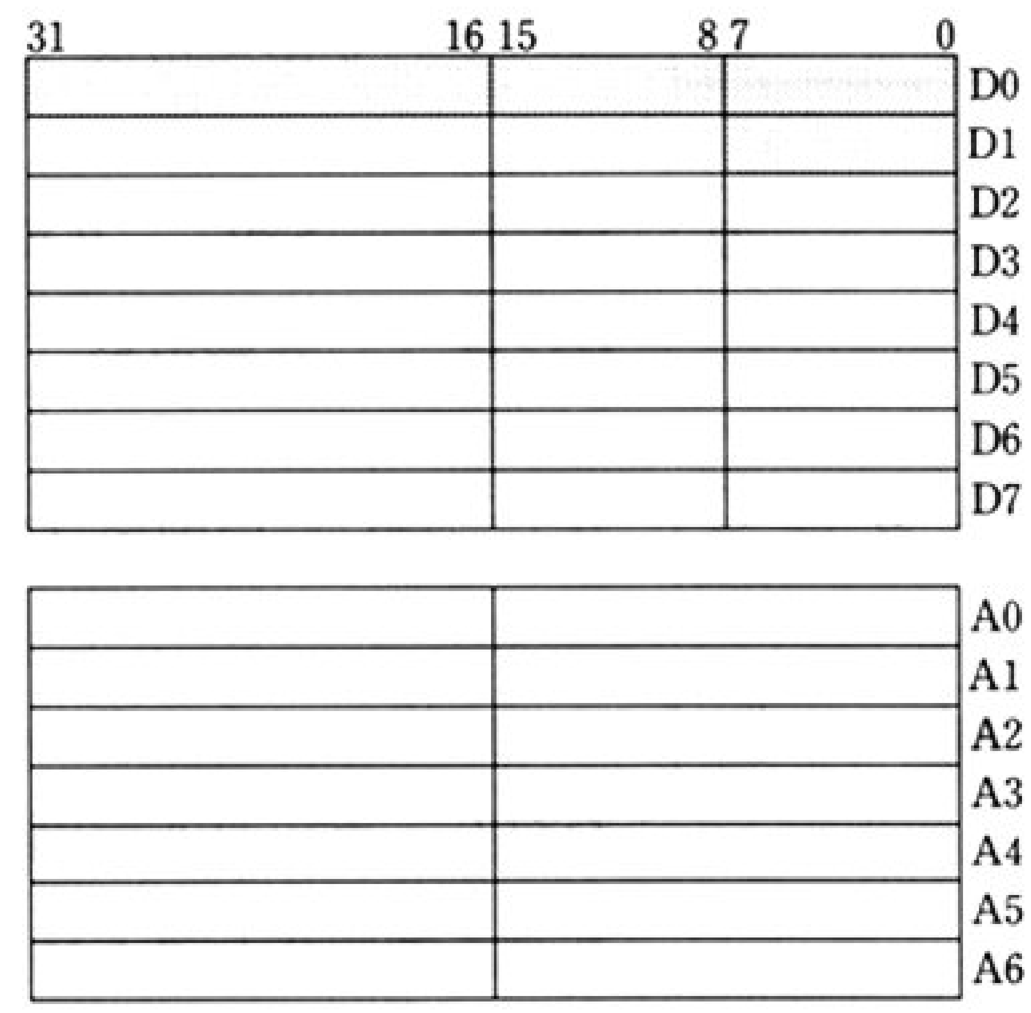
        moveq.l #_B_LEFT,d0     * IOCSコール番号をセット
        move.b  10(a6),d1       * 移動桁数セット
        trap    #iocs           * IOCSコール

        movem.l (sp)+,d0-d1     * d0 復帰
        unlk    a6
        rtr                     * 呼び出し元に帰ります
        .end
```

B_CLR_ST

● B_CLR_ST　　IOCS コール番号：$2A

機　　能　画面の複数行消去



入　　力　D0. L　　$2A　　IOCS コール番号
D1. B　　画面の消去指定

リターン　なし（D0. L の内容は保証されません）

解　　説　画面の複数行を消去します。
入力の D1. B は画面の消去方法を指定します。
内容は次の通りです。

D1. B ＝ 0……カーソル位置から最終行右端まで消去する
D1. B ＝ 1……先頭行左端からカーソル位置まで消去する
D1. B ＝ 2……画面全体を消去しカーソルは先頭行左端に移動する

プログラム例　画面を消去します。

```
********************************************
* B_CLR_ST                              $2A
********************************************

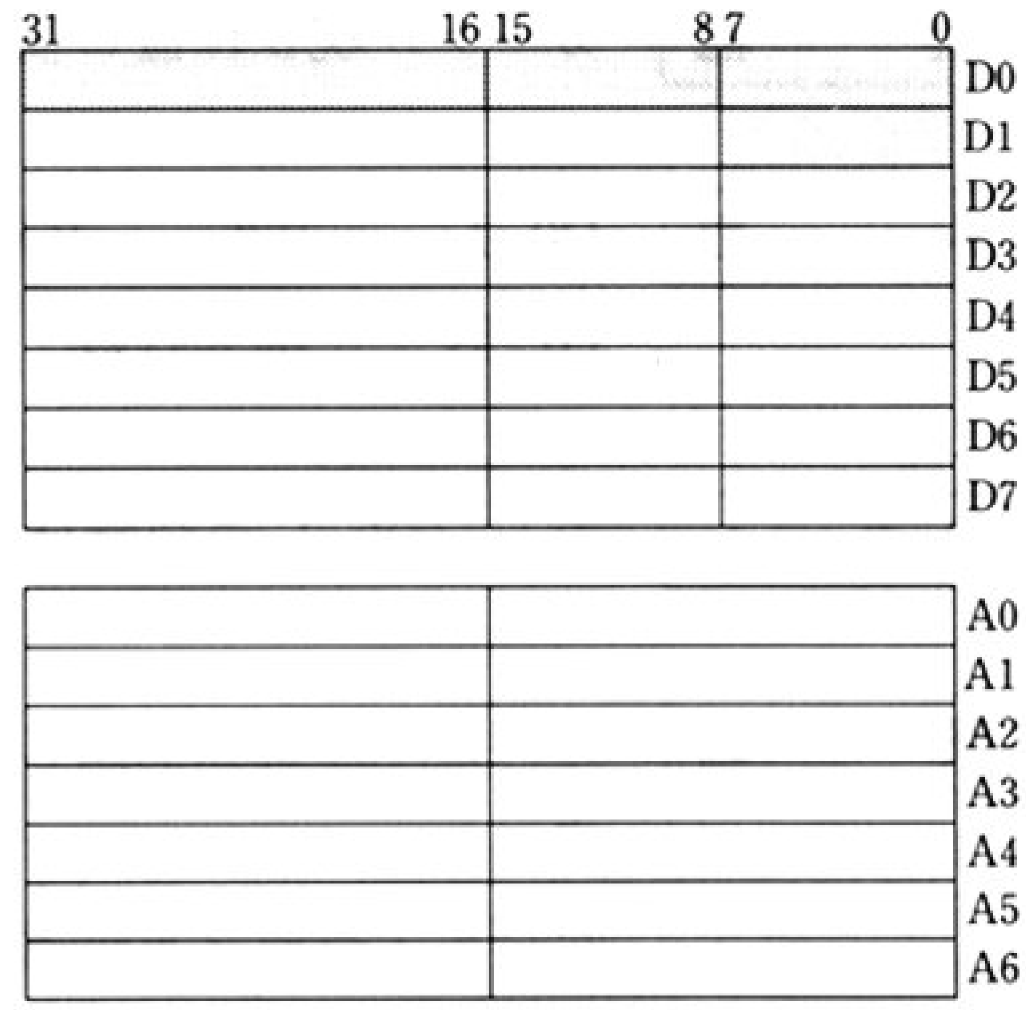
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_CLR_ST,d0       * IOCSコール番号をセット
        move.b  #2,d1               * 画面全体消去を指定
        trap    #iocs               * IOCSコール
        .
        .
        .end
```

B_ERA_ST

● B_ERA_ST　　IOCS コール番号：$2B



機能　現在行の複数桁消去

入力　D0. L　$2B　IOCS コール番号
D1. B　行の消去指定

リターン　なし（D0. L の内容は保証されません）

解説　現在行の複数桁を消去します。
入力の D1. B は、行の消去方法を指定します。
内容は次の通りです。

D1. B ＝ 0……カーソル位置から行の右端まで消去する
D1. B ＝ 1……行の左端からカーソル位置まで消去する
D1. B ＝ 2……カーソルのある 1 行だけを消去する

プログラム例　カーソル行を消去します。

```
********************************************
* B_ERA_ST                              $2B
********************************************

        .include        iocscall.mac
        .include        doscall.mac

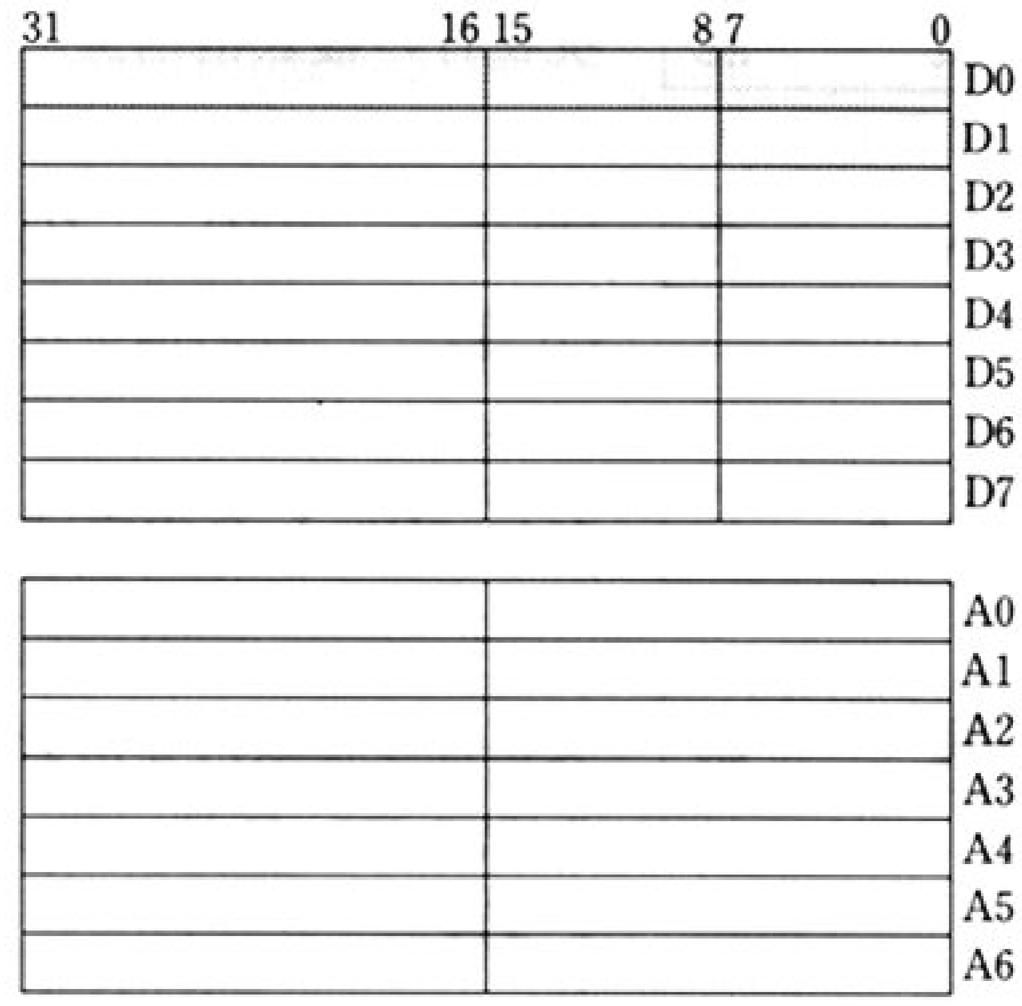
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_ERA_ST,d0     * IOCSコール番号をセット
        move.b  #2,d1             * カーソル行全体消去を指定
        trap    #iocs             * IOCSコール
        .
        .
        .end
```

B_INS

● B_INS　　　IOCSコール番号：$2C

機　能　カーソル現在位置に複数行挿入



入　力　D0. L　　$2C　　IOCSコール番号
D1. B　　挿入行数

リターン　なし (D0. Lの内容は保証されません)

解　説　カーソルの現在位置に、複数行挿入します。
ただし、カーソル位置の桁数は0になります。
入力のD1. Bは、挿入行数を指定します。
0を指定した場合は、1と見なされます。

プログラム例　カーソル現在位置に1行挿入します。

```
*********************************************
* B_INS                                   $2C
*********************************************

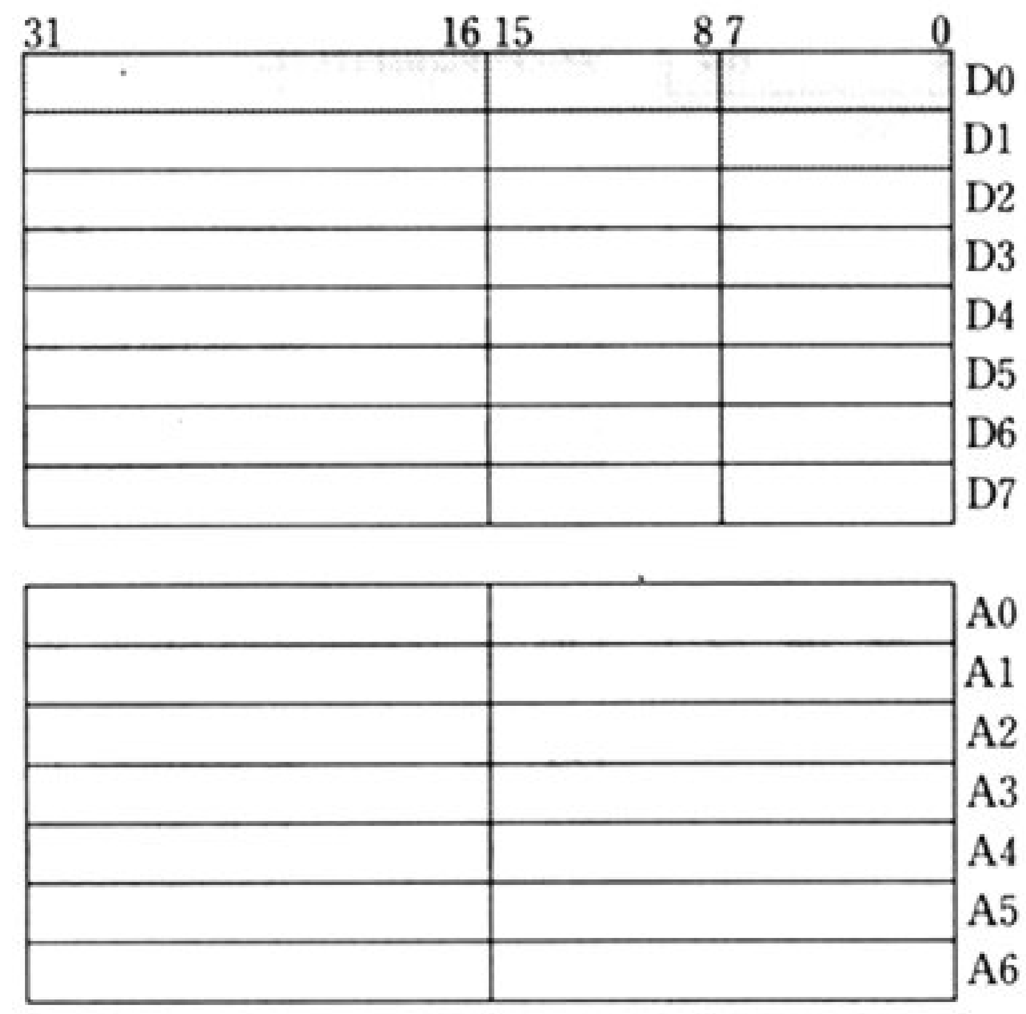
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_INS,d0      * IOCSコール番号をセット
        move.b  #1,d1           * 1 行挿入指定
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_DEL

● B_DEL　　IOCS コール番号：$2D



機　能　カーソル現在位置から複数行削除

入　力　D0. L　$2D　IOCS コール番号
D1. B　削除行数

リターン　なし（D0. L の内容は保証されません）

解　説　カーソルの現在位置から、複数行削除します。
入力の D1. B は、削除行数を指定します。
0 を指定した場合は、1 と見なされます。

プログラム例　カーソル現在位置から 3 行削除します。

```
*******************************************
* B_DEL                                 $2D
*******************************************

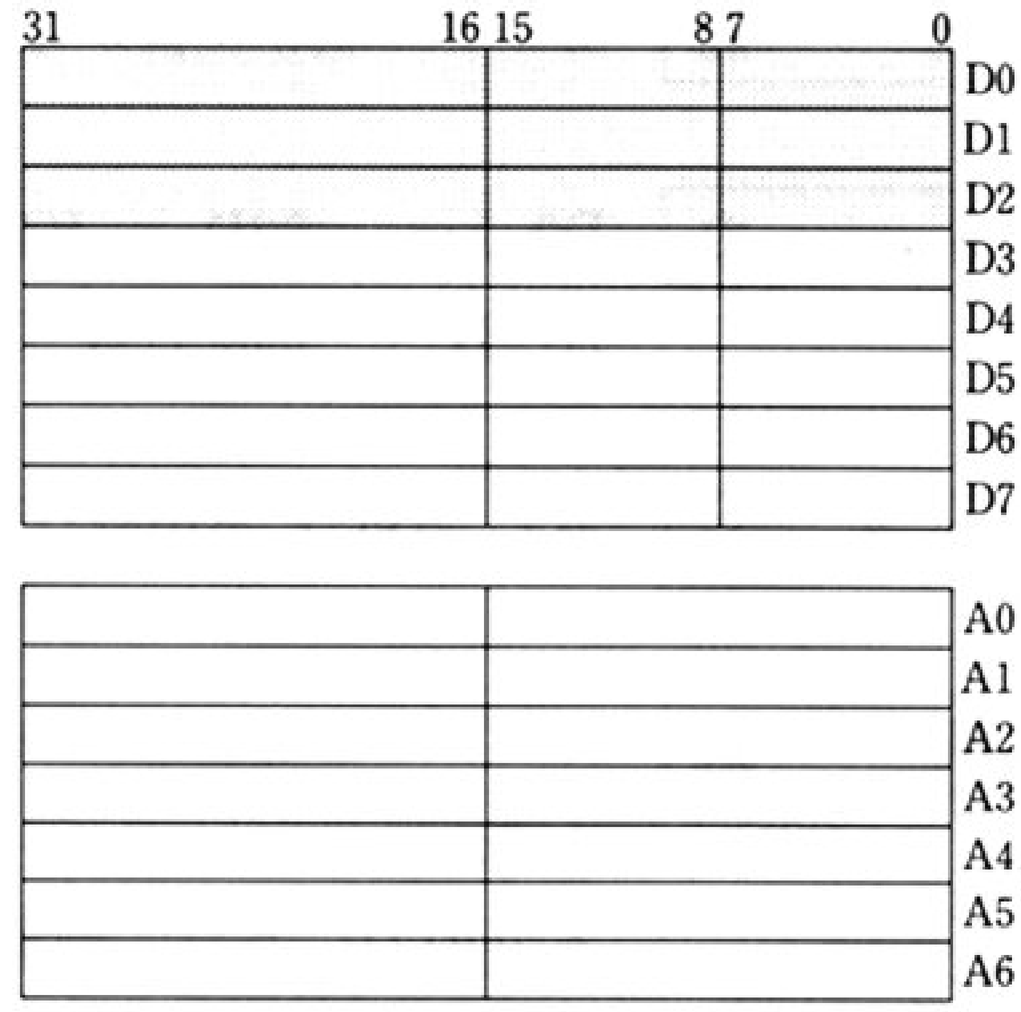
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_DEL,d0      * IOCSコール番号をセット
        move.b  #3,d1           * 3 行削除指定
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_CONSOL

● B_CONSOL　　IOCS コール番号：$2E



機　能	表示範囲指定
入　力	D0. L　$2E　IOCS コール番号 D1. L　表示開始ドット位置 D2. L　表示範囲指定
リターン	D0. L の内容は保証されません。 D1. L　変化前のドット位置 D2. L　変化前の範囲

解　説

表示範囲を指定します。
ただし、カーソルは先頭行左端に移動します。
X68000 は、1024×1024 ドットのテキスト画面（実画面エリア）を 4 プレーン持っています。
実際に CRT に表示されるテキスト画面（表示画面）は、最大で 768×512 ドットです。
したがって、8×16 ドットの半角フォントを使用した場合、96 桁×32 行のテキストを表示できることになります。
この IOCS コールではテキスト表示範囲を仮想画面における表示画面の左上のドット座標と、表示画面の桁数、行数で指定します。
入力の D1. L の上位 16 ビット、下位 16 ビットは、それぞれ横方向の表示開始ドット位置、縦方向の表示開始ドット位置です。
ただし、D1. L が−1 の場合、前のままで変化させないことを意味します。
入力の D2. L の上位 16 ビット、下位 16 ビットはそれぞれ（横方向の表示桁数）−1、（縦方向の表示行数）−1 です。
ただし、D2. L が−1 の場合、前のままで変化させないことを意味します。
変化前の値が D1. L、D2. L に返されます。
リターンの D1. L の上位 16 ビット、下位 16 ビットは、それぞれ変化前の横方向の表示開始ドット位置、縦方向の表示開始ドット位置です。
リターンの D2. L の上位 16 ビット、下位 16 ビットは、それぞれ変化前の（横方向の表示桁数）−1、（縦方向の表示行数）−1 です。
入力の横方向の表示開始ドット位置は、0〜1008 までの 16 の倍数でなければなりません。
入力の縦方向の表示開始ドット位置は、0〜1020 までの 4 の倍数でなければなりません。

入力、リターンともに（横方向の表示桁数）−1は、0〜127までの桁数でなければなりません。
（縦方向の表示行数）−1は、0〜63までの行数でなければなりません。

プログラム例

```
*********************************************
* B_CONSOL                                $2E
*********************************************

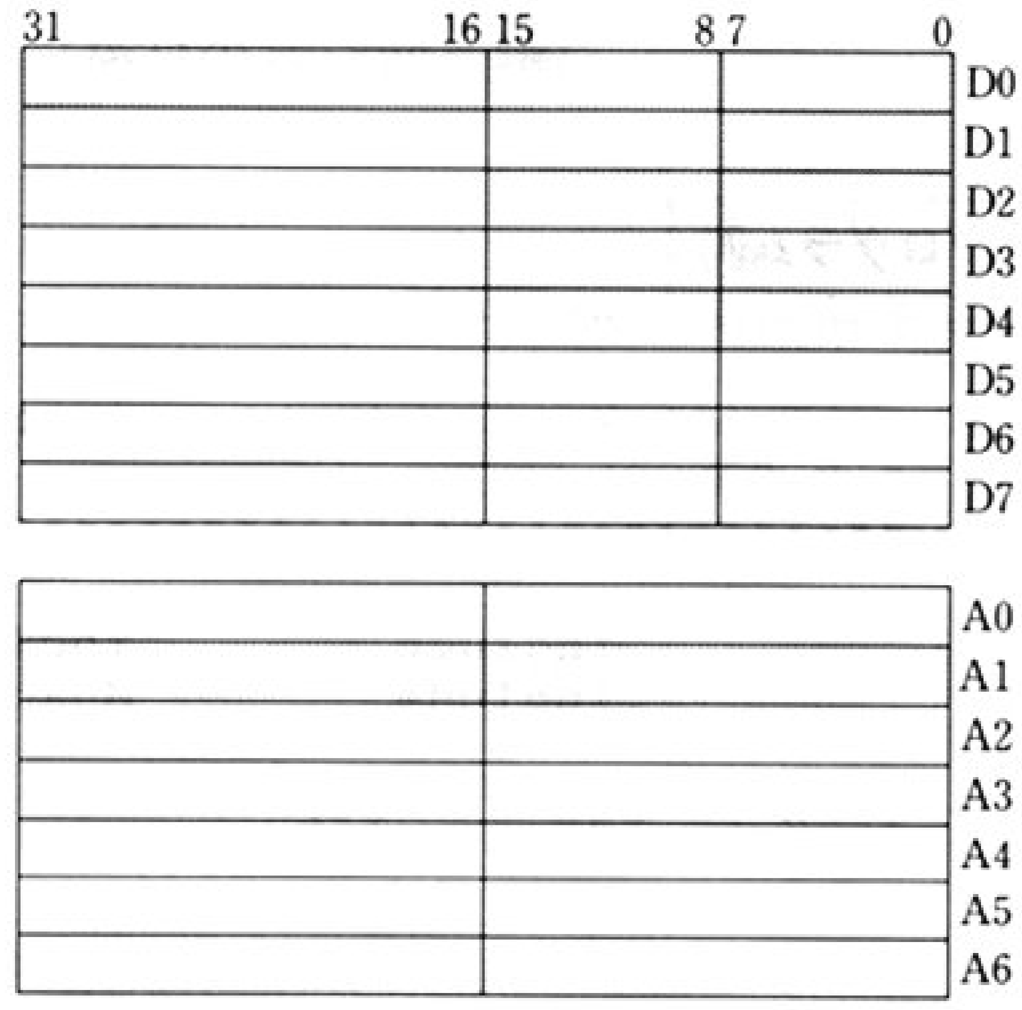
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        moveq.l #_B_CONSOL,d0   * IOCSコール番号をセット
        move.l  #$0010_0020,d1  * d1.l = x * 16 * $10000 + y * 16;
                                * x = 1, y = 2
        move.l  #$004F_0018,d2  * d2.l = c * $10000 + L;
                                * c = 79, l = 24
        trap    #iocs           * IOCSコール
        dc.w    _EXIT           * プログラムの終了
        .end
```

B_PUTMES

● B_PUTMES　　IOCS コール番号：$2F



機　能　絶対座標位置から指定文字列表示

入　力

D0. L　$2F　IOCS コール番号
D1. B　属性
D2. W　絶対座標桁位置
D3. W　絶対座標行位置
D4. W　(表示桁数)－1
A1. L　文字列先頭アドレス

リターン

D2. W　次の桁位置
A1. L　文字列の終端 (ヌル文字) の次のアドレス
D0. L　不定 (内容は保証されません)
D4. W　不定 (内容は保証されません)

解　説

この機能はファンクションキー行表示用です。
指定した絶対座標位置から、指定文字列を表示します。
また、B_CONSOL ($2E；表示範囲を指定する) の影響を受けません。
桁数以上は表示しませんが、漢字の第 2 バイト目が余るような場合、その漢字は表示されずスペースが表示されます。
途中で文字列が終わりになったら、後はスペースで埋めます。
入力の D1. B は属性の指定で、内容は次の通りです。

ビット3	ビット2	ビット1	ビット0	意　味
0	0	0	0	黒
0	0	0	1	水色
0	0	1	0	黄色
0	0	1	1	白
0	1	0	0	黒
0	1	0	1	水色・強調
0	1	1	0	黄色・強調
0	1	1	1	白・強調
1	0	0	0	黒
1	0	0	1	水色・リバース

1	0	1	0	黄色・リバース
1	0	1	1	白・リバース
1	1	0	0	黒
1	1	0	1	水色・強調・リバース
1	1	1	0	黄色・強調・リバース
1	1	1	1	白・強調・リバース

入力のD2. W、D3. Wはそれぞれ絶対座標の桁位置、行位置です。
入力のD4. Wは（表示桁数）−1です。
入力のA1. Lは文字列の先頭アドレスです。
この文字列はヌル文字（0）で終わらなければなりません。
リターンのD2. WとA1. Lは、それぞれ次の桁位置と文字列の終端（ヌル文字）の次のアドレスです。
また、リターンのD0. L、D4. Wの内容は保証されません。

プログラム例

```
*******************************************
* B_PUTMES                             $2F
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

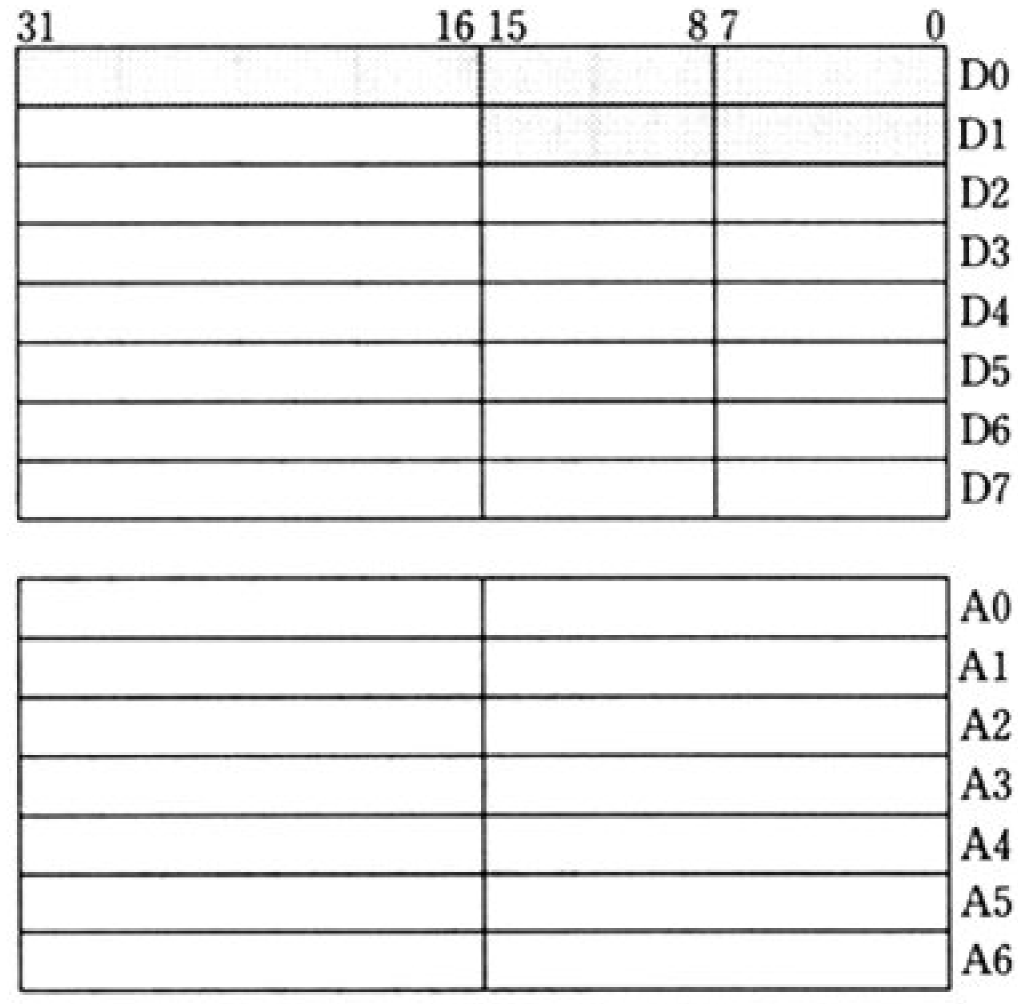
iocs    equ     $0f

        .text
        moveq.l #_B_PUTMES,d0    * IOCSコール番号をセット
        move.b  #%1101,d1        * 属性(リバース、強調、水色)
        move.w  #16,d2           * 桁位置 16
        move.w  #20,d3           * 行位置 20
        move.w  #10,d4           * 表示桁数 10
        lea     w_msg,a1         * 表示文字列先頭アドレス
        trap    #iocs            * IOCSコール
        dc.w    _EXIT            * プログラムの終了

        .data
w_msg:  dc.b    " Hello",0
        .end
```

SET232C

● SET232C　　　IOCS コール番号：$30



機　能　RS-232C 設定

入　力　D0. L　$30　IOCS コール番号
D1. W　RS-232C の設定値

リターン　D0. L　前のモード

解　説　RS-232C のモードを設定します。
入力の D1. W は、RS-232C のモードを設定します。
D1. W に−1を指定すると、前のモードを返すだけです。
設定値の内容は次の通りです。

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
D1. W	ST		PP		BB		X	0	0	0	0	0	0	BAU		

ボーレート指定
000：75
001：150
010：300
011：600
100：1200
101：2400
110：4800
111：9600

XON/XOFF 処理
0：処理しない
1：処理する

ビット長
00：5 ビット以下
01：6 ビット
10：7 ビット
11：8 ビット

パリティ
01　：奇数
11　：偶数
00、10：パリティなし

ストップビット
01　：1
10　：1.5
00、11：2

プログラム例 RS-232C 通信パラメータを設定します。

```
********************************************
* SET232C                                $30
********************************************

        .include        iocscall.mac
        .include        doscall.mac

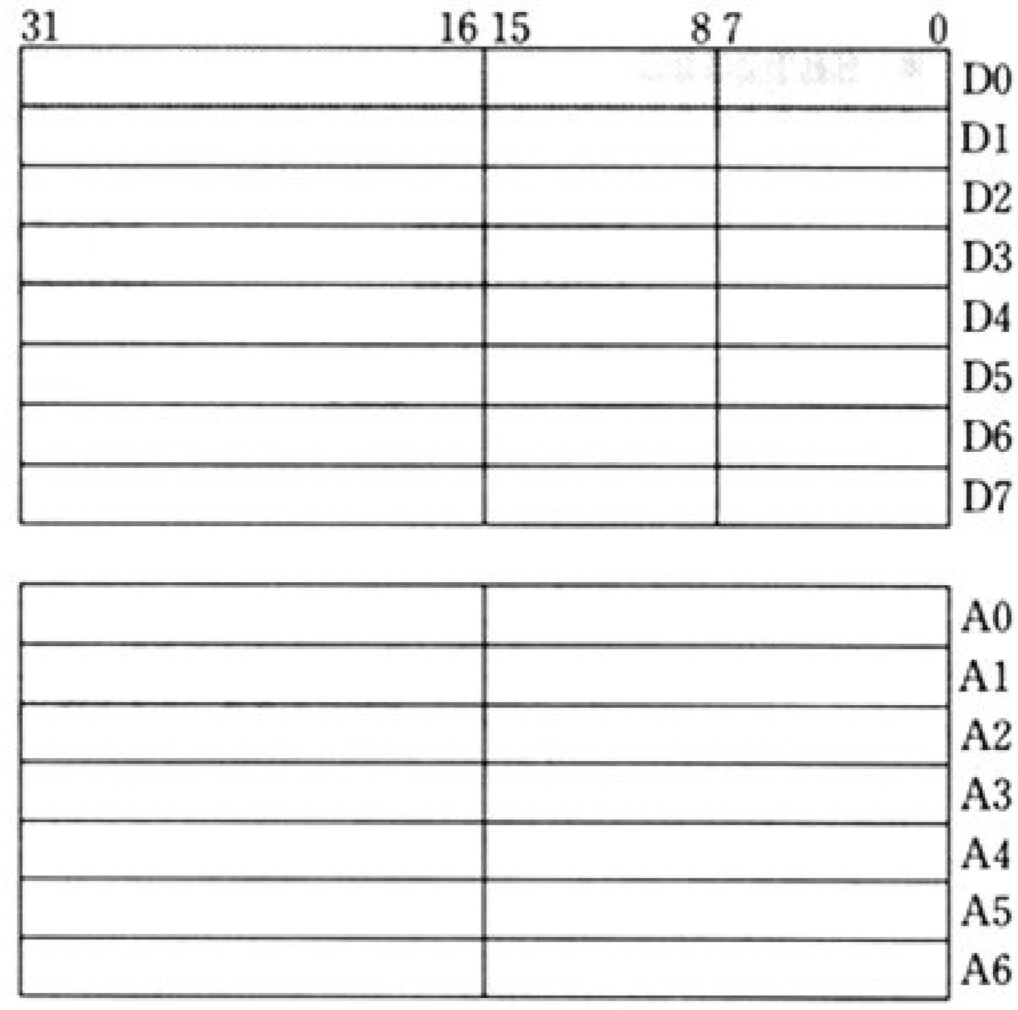
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SET232C,d0            * IOCSコール番号をセット
        move.w  #%0100_1110_0000_0100,d1* stop:          1
                                        * parity:        なし
                                        * bit:           8
                                        * xon/off:       する
                                        * si/so:         しない
                                        * bps:           1200
        trap    #iocs                   * IOCSコール
        .
        .
        .end
```

LOF232C

● LOF232C　　　IOCS コール番号：$31

機　能	RS-232C 受信バッファ内データ数取得
入　力	D0. L　　$31　　IOCS コール番号
リターン	D0. W　　データ数
解　説	RS-232C の受信バッファ内のデータ数を得ます。 リターンの D0. W は、RS-232C の受信バッファ内のデータ数です。



プログラム例　RS-232C 受信バッファ内のデータ数を調べ、必要な処理に分岐します。

```
********************************************
* LOF232C                              $31
********************************************

        .include        iocscall.mac
        .include        doscall.mac

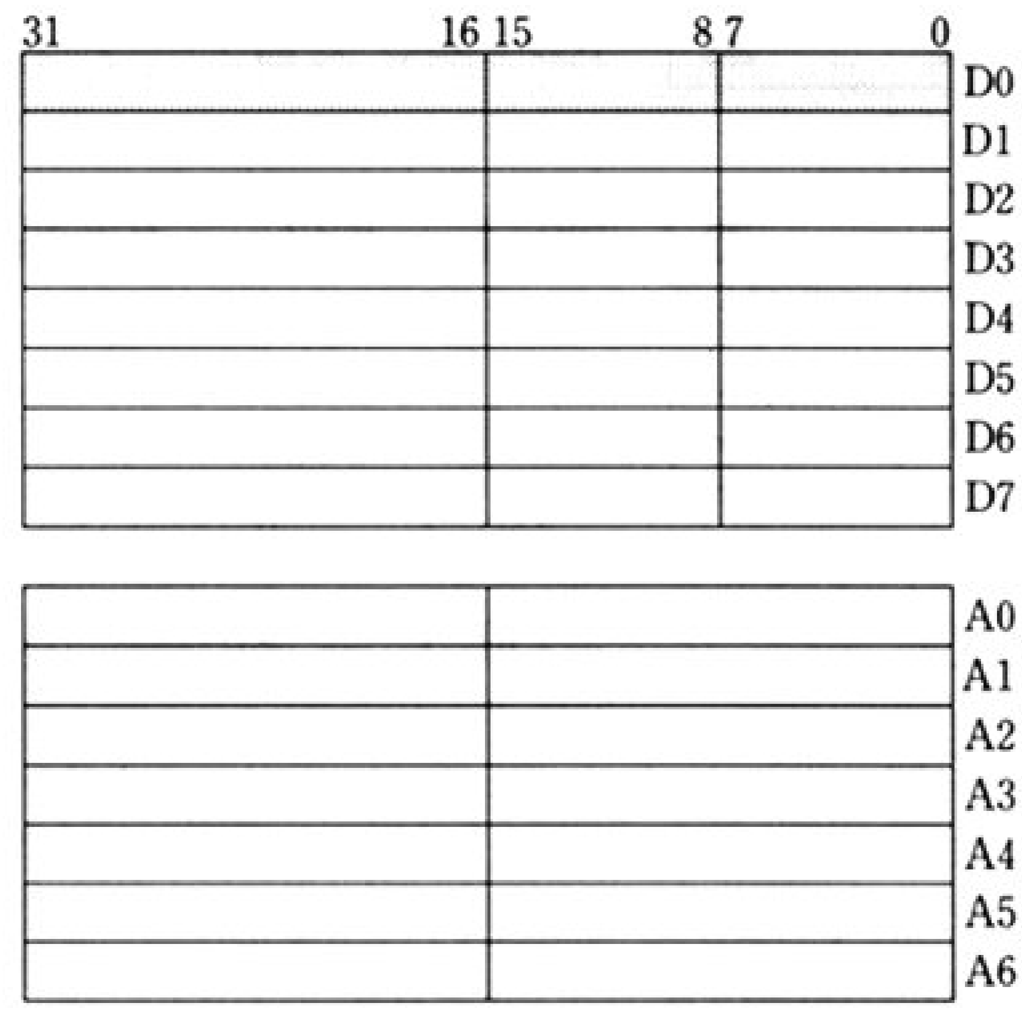
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_LOF232C,d0      * IOCSコール番号をセット
        trap    #iocs             * IOCSコール
        tst.w   d0                * バッファ内にデータがあれば
        beq     LOF0              * LOF0 へブランチ(受信データ処理)
        rts
LOF0:
        bsr     get232c
        rts
        .
        .
get232c:
        .
        .
        rts
        .
        .
        .end
```

INP232C

● INP232C　　　IOCS コール番号：$32

機　　能	RS-232C 受信データ取得
入　　力	D0. L　　$32　　IOCS コール番号
リターン	D0. W　　受信データ
解　　説	RS-232C の受信データを得ます。 リターンの D0. W の下位 8 ビットの内容が RS-232C の受信データです。
プログラム例	受信バッファから 1 文字読み込みます。




```
******************************************
* INP232C                             $32
******************************************

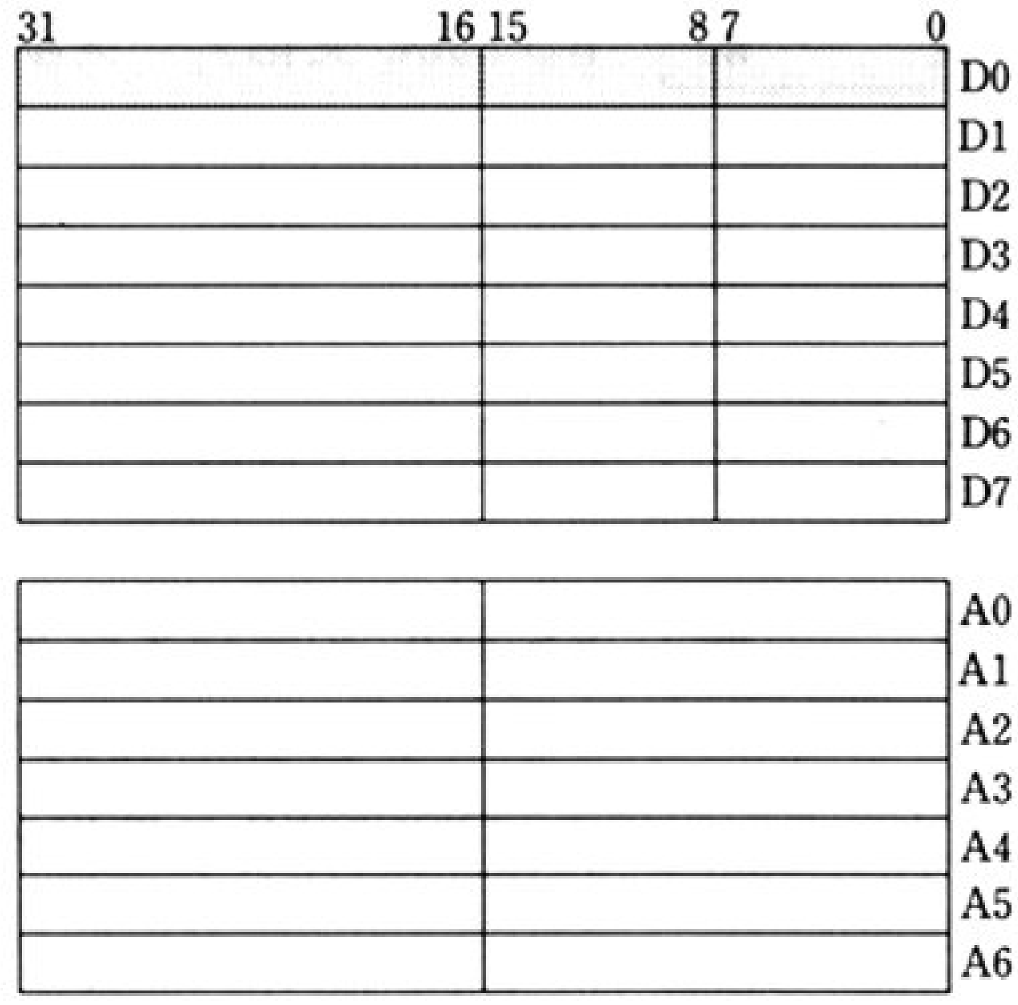
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_INP232C,d0     * IOCSコール番号をセット
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

ISNS232C

● ISNS232C　　IOCS コール番号：$33



機　能　RS-232C 受信データチェック

入　力　D0. L　$33　IOCS コール番号

リターン　D0. L　受信データ

D0. L ＝ 0 …………受信データなし

D0. L ＝ $0100?? ……受信データあり（$??は受信データ）

解　説　RS-232C の受信データをチェックします。

プログラム例　受信バッファを調べます。

このIOCS コールで次に受信バッファから読み込まれるデータの種類を前もって調べて、漢字やエスケープシーケンスなどに対応した受信ルーチンに分岐すれば、通信プログラムの構造が分かりやすくします。

```
*******************************************
* ISNS232C                             $33
*******************************************

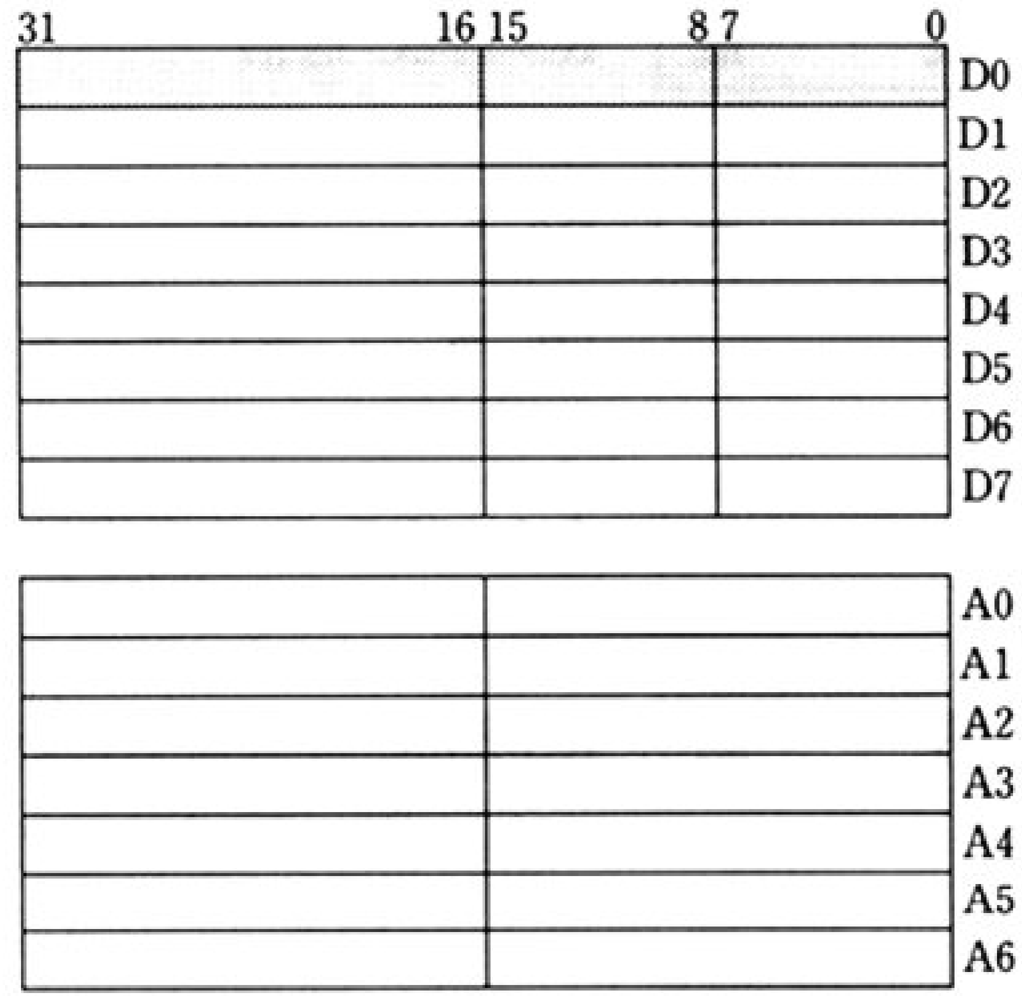
         .include        iocscall.mac
         .include        doscall.mac

iocs     equ      $0f

         .text
         .
         .
loop:
         moveq.l #_ISNS232C,d0    * IOCSコール番号をセット
         trap    #iocs            * IOCSコール
         tst.l   d0
         beq     loop
         andi.l  #$00ff,d0
         cmp.b   #$1b,d0
         .
         .
         .end
```

OSNS232C

● OSNS232C　　IOCS コール番号：$34



機　　能　RS-232C 送信可能状態チェック

入　　力　D0. L　$34　IOCS コール番号

リターン　D0. L　送信可／不可

D0. L ＝ 0…送信できない

D0. L ＝ 4…送信できる（バッファは空で、XON でもない）

解　　説　RS-232C に送信が可能かどうかチェックします。

プログラム例

```
********************************************
* OSNS232C                              $34
********************************************

        .include        iocscall.mac
        .include        doscall.mac

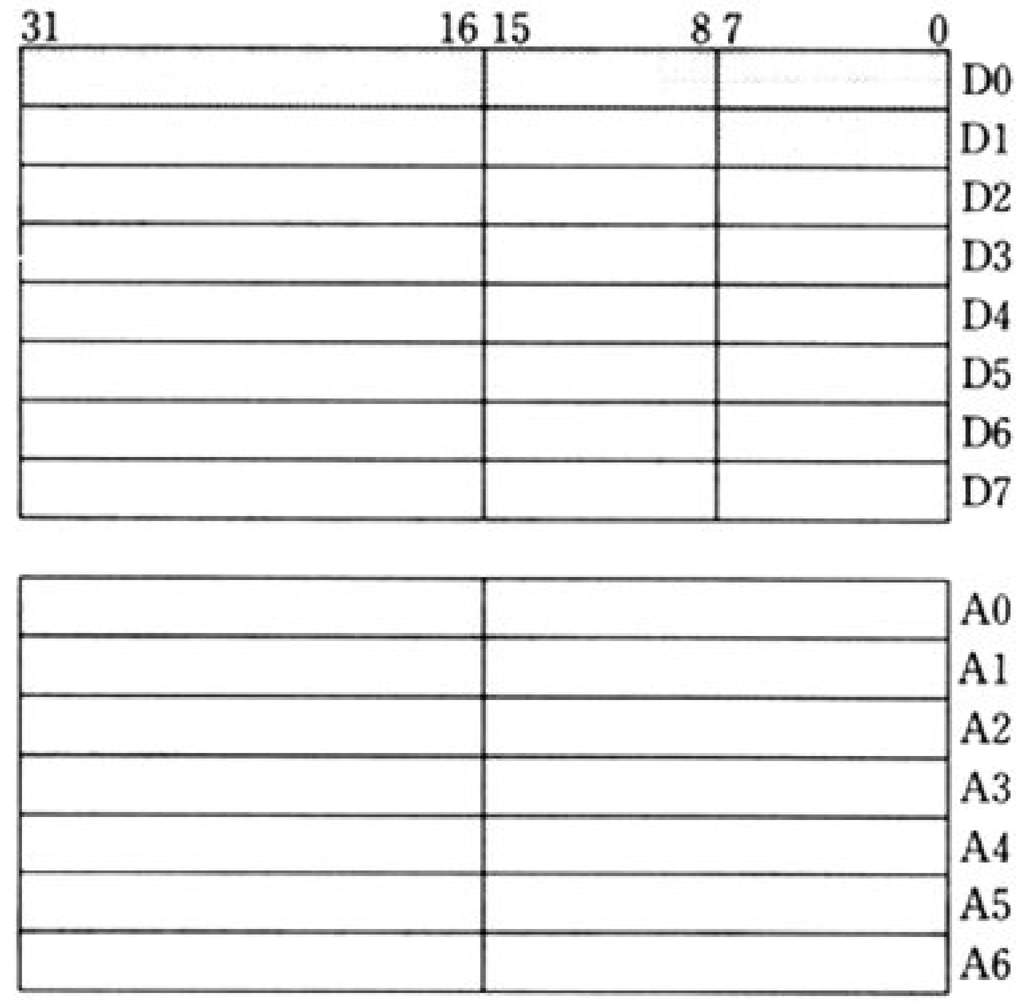
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OSNS232C,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

OUT232C

● OUT232C　　IOCS コール番号：$35

機　　能	RS-232C 送信
入　　力	D0. L　$35　IOCS コール番号 D1. B　送信データ
リターン	なし（D0. L の内容は保証されません）
解　　説	RS-232C に送信を行います。 入力の D1. B は送信データです。



プログラム例

```
*******************************************
* OUT232C                             $35
*******************************************

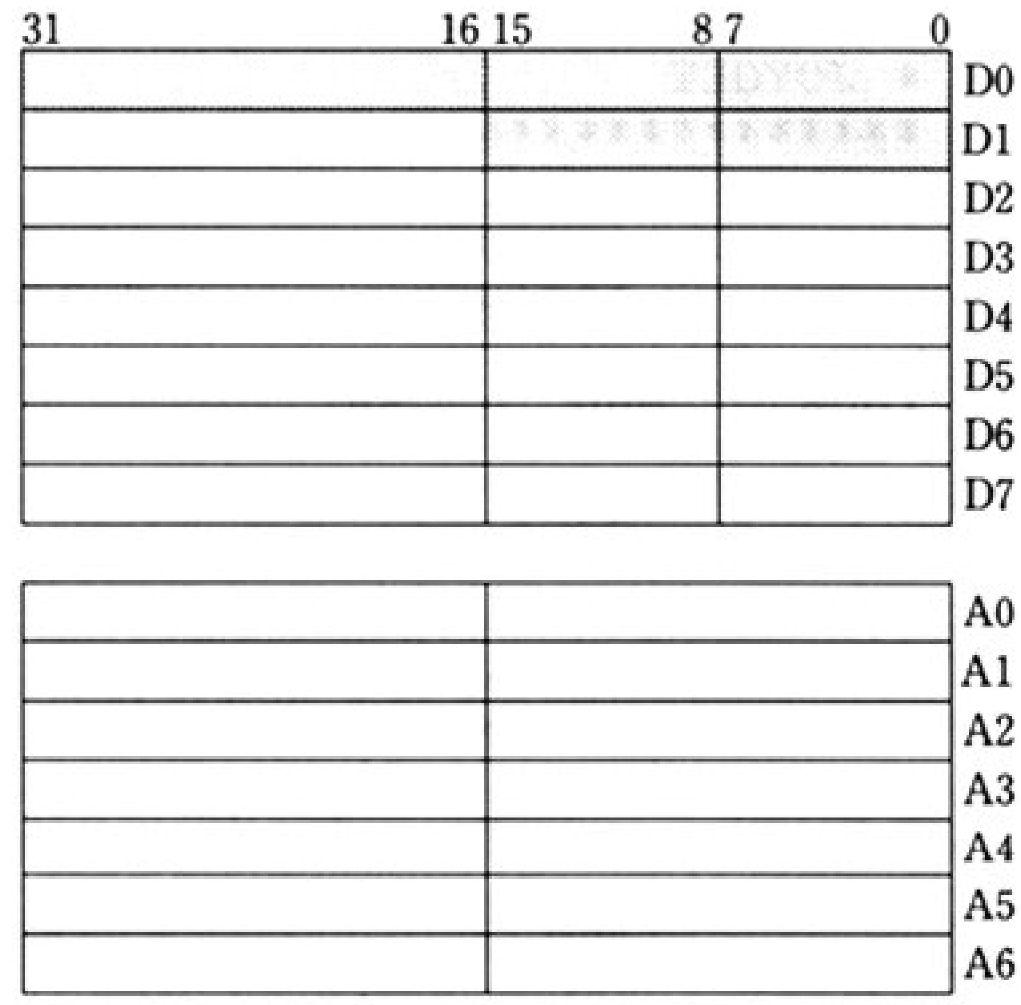
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OUT232C,d0     * IOCSコール番号をセット
        move.b  (a0)+,d1         * a0 でポイントされるデータを送ります
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

JOYGET

● JOYGET　　IOCS コール番号：$3B

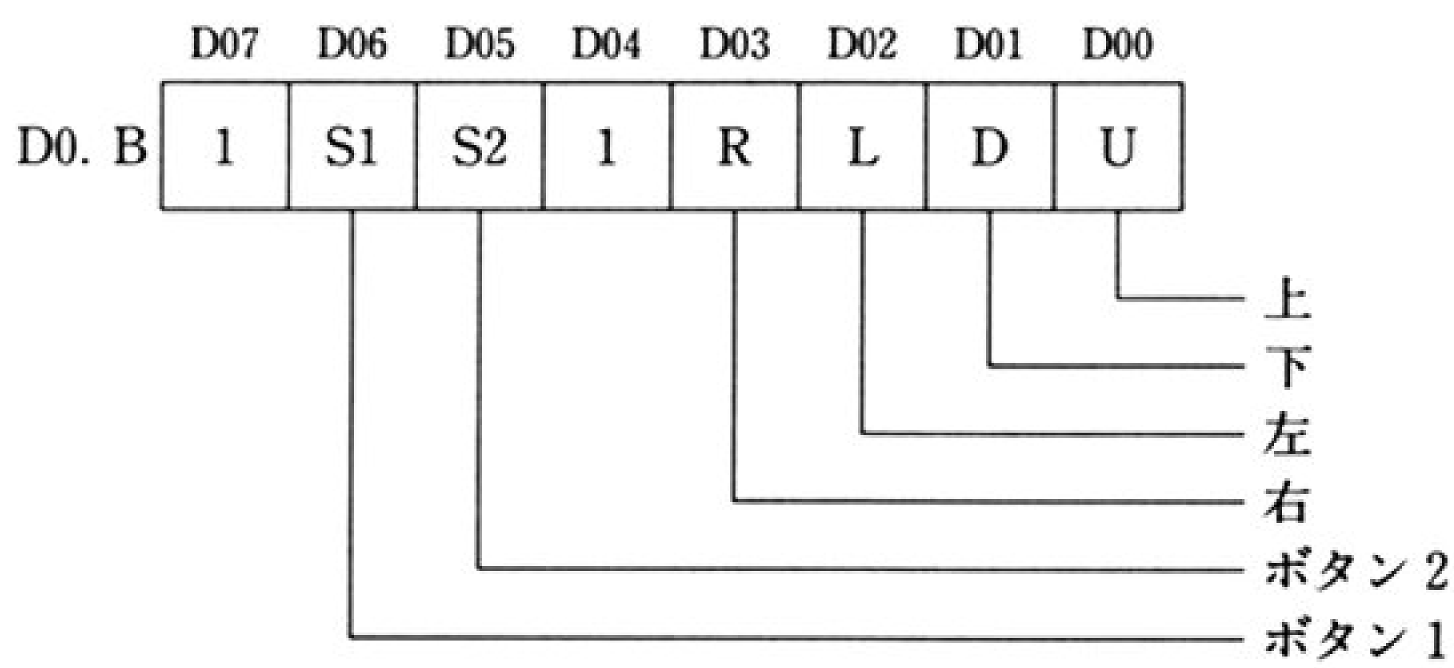


機　　能　ジョイスティックデータ読み込み

入　　力　D0. L　$3B　IOCS コール番号
D1. W　ジョイスティックの番号

リターン　D0. B　ジョイスティックのデータ

解　　説　ジョイスティックのデータを読み込みます。
入力の D1. W は選択するジョイスティック番号で、内容は次の通りです。
D1. W＝0……ジョイスティック 1
D1. W＝1……ジョイスティック 2
リターンの D0. B は、ジョイスティックのデータで、内容は次の通りです。



押されていれば 0、離されていれば 1 になります。

プログラム例

```
*******************************************
* JOYGET                              $3B
*******************************************

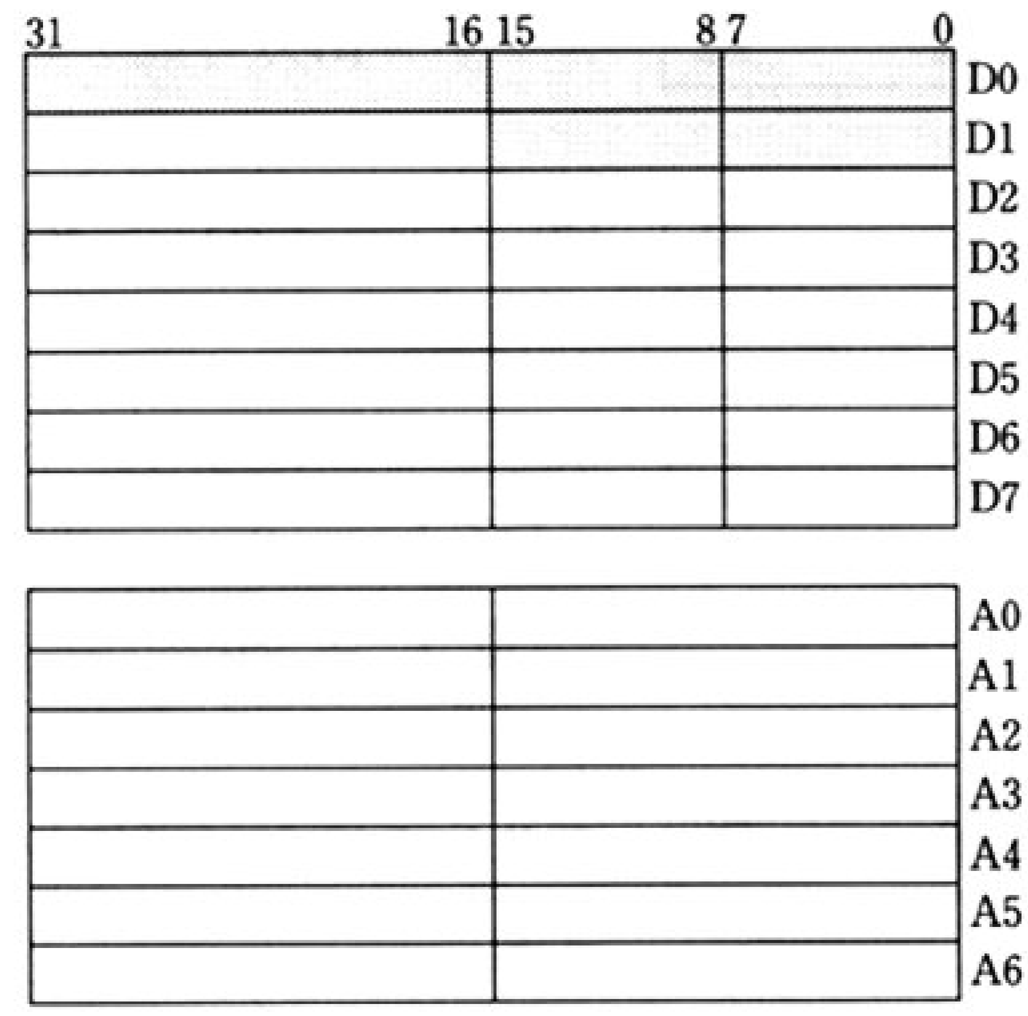
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_JOYGET,d0     * IOCSコール番号をセット
        move.w  #0,d1           * ジョイスティック1を選択
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

INIT_PRN

● INIT_PRN　　IOCS コール番号：$3C



機　能　プリンタポート初期化

入　力　D0. L　$3C　IOCS コール番号
D1. W　プリントフォーマット指定

リターン　D0. L　出力可／不可
D0. L ＝ 0 ……出力不可
D0. L ＝ 20……出力可能

解　説　プリンタポートを初期化します。
入力の D1. W は、OUTPRN（$3F）用のプリントフォーマットを指定します。
上位 8 ビットは、(1 ページの行数)－1 を指定します。
ここに$FF を指定すると、ページ指定なしになります。
下位 8 ビットは、(1 行の文字数)－1 を指定します。
ここに$FF を指定すると、行指定なしになります。

プログラム例

```
********************************************
* INIT_PRN                             $3C
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_INIT_PRN,d0    * IOCSコール番号をセット
        move.w  #63*256+79,d1    * 行数 64、桁数 80
        trap    #iocs            * IOCSコール
        cmp.l   #$20,d0          * d0.l <> $20 のとき
        bne     error            *       エラー処理へ
        rts                      * 正常終了

*---[ Error ]---
error:
        .
        .
        rts
        .
        .
        .end
```

SNSPRN

● SNSPRN　　IOCS コール番号：$3D

機　　能　プリンタ出力可能チェック

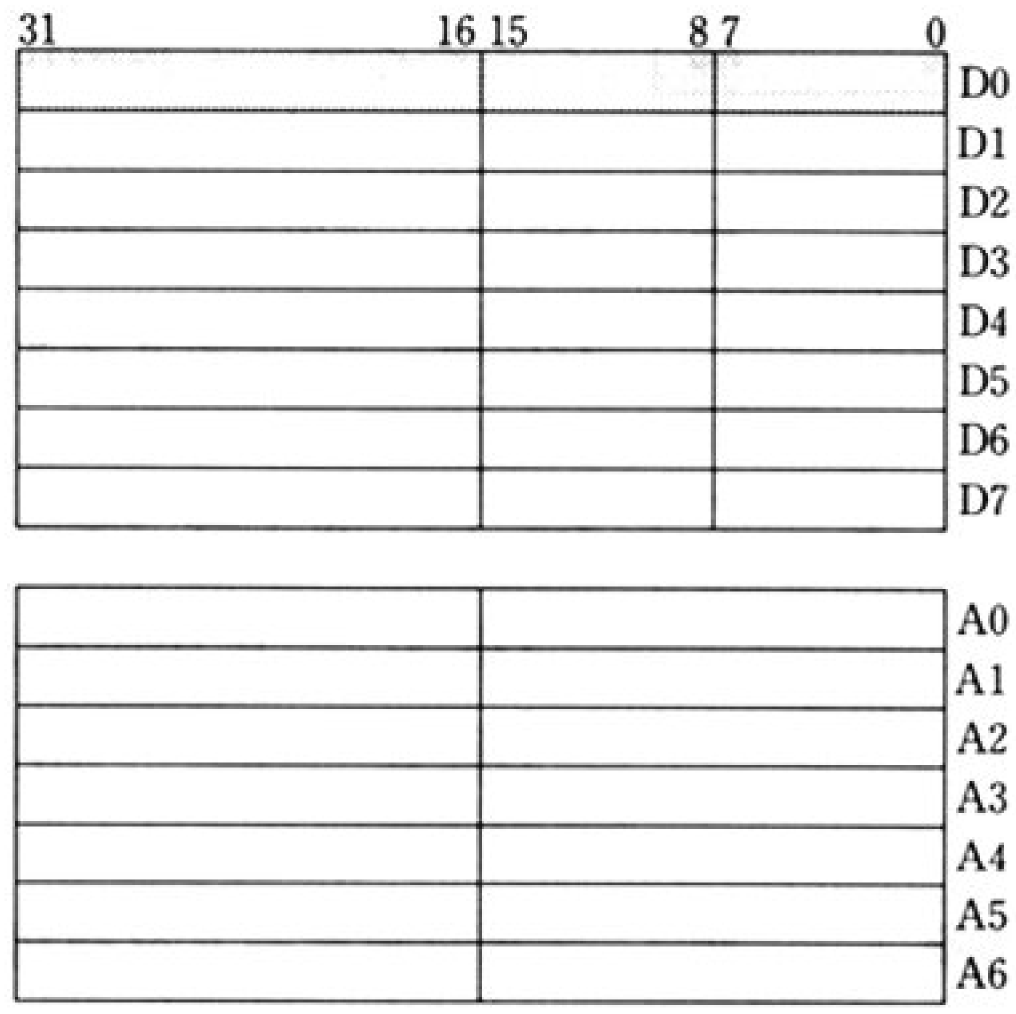
入　　力　D0. L　　$3D　　IOCS コール番号

リターン　D0. L　　出力可／不可

D0. L ＝ 0 ……出力不可

D0. L ＝ 20……出力可能

解　　説　プリンタ出力が可能かどうかチェックします。



プログラム例

```
*******************************************
* SNSPRN                               $3D
*******************************************

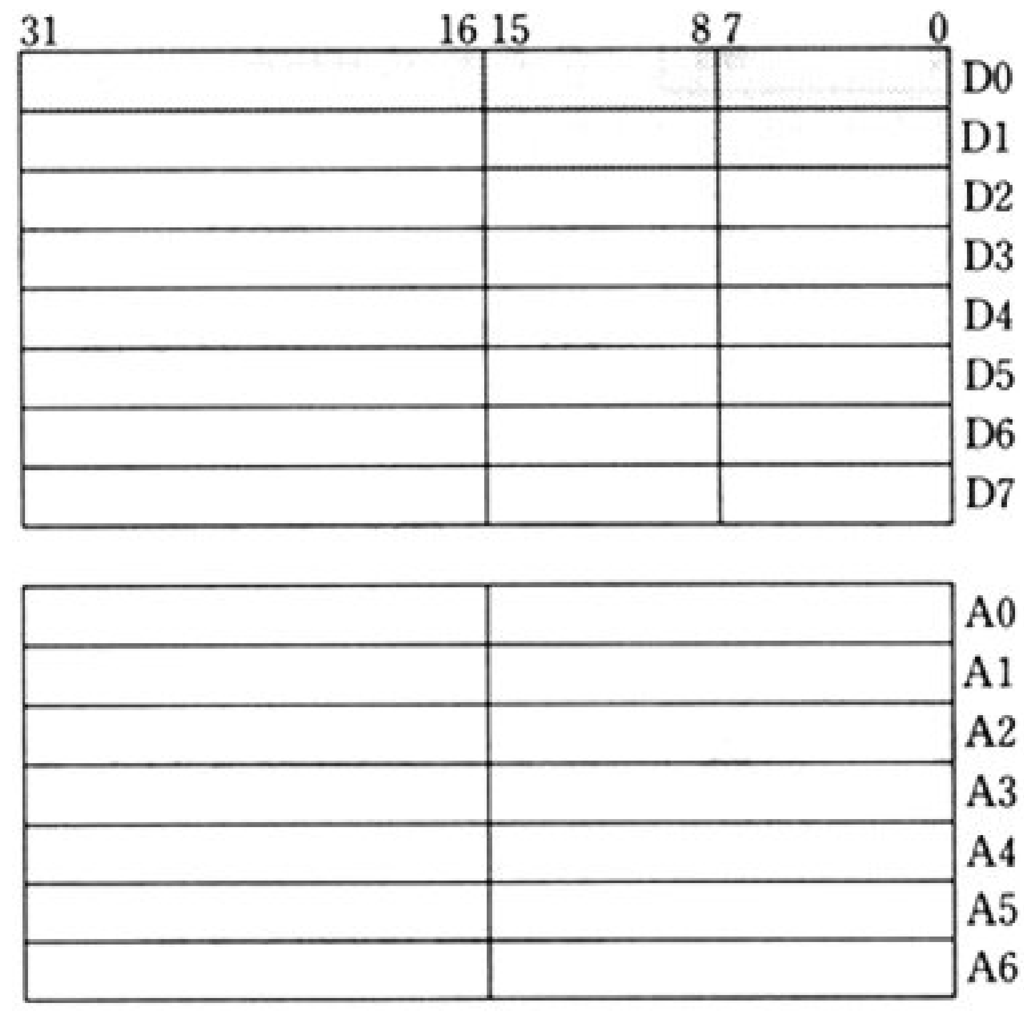
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SNSPRN,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        tst.l   d0
        beq     error
        .
        .
*---[ Error ]---
error:
        .
        .
        .end
```

OUTLPT

● OUTLPT　　IOCS コール番号：$3E



機能　プリンタ直接出力

入力　D0. L　$3E　IOCS コール番号
D1. W　出力データ

リターン　なし（D0. L の内容は保証されません）

解説　プリンタに直接データを出力します。
入力の D1. W で出力データを指定します。
D1. W の上位バイトは無視されます。
漢字処理せずに指定データを直接プリンタに出力します。
また、自動改行、自動改頁しません。

プログラム例

```
*********************************************
* OUTLPT                                  $3E
*********************************************

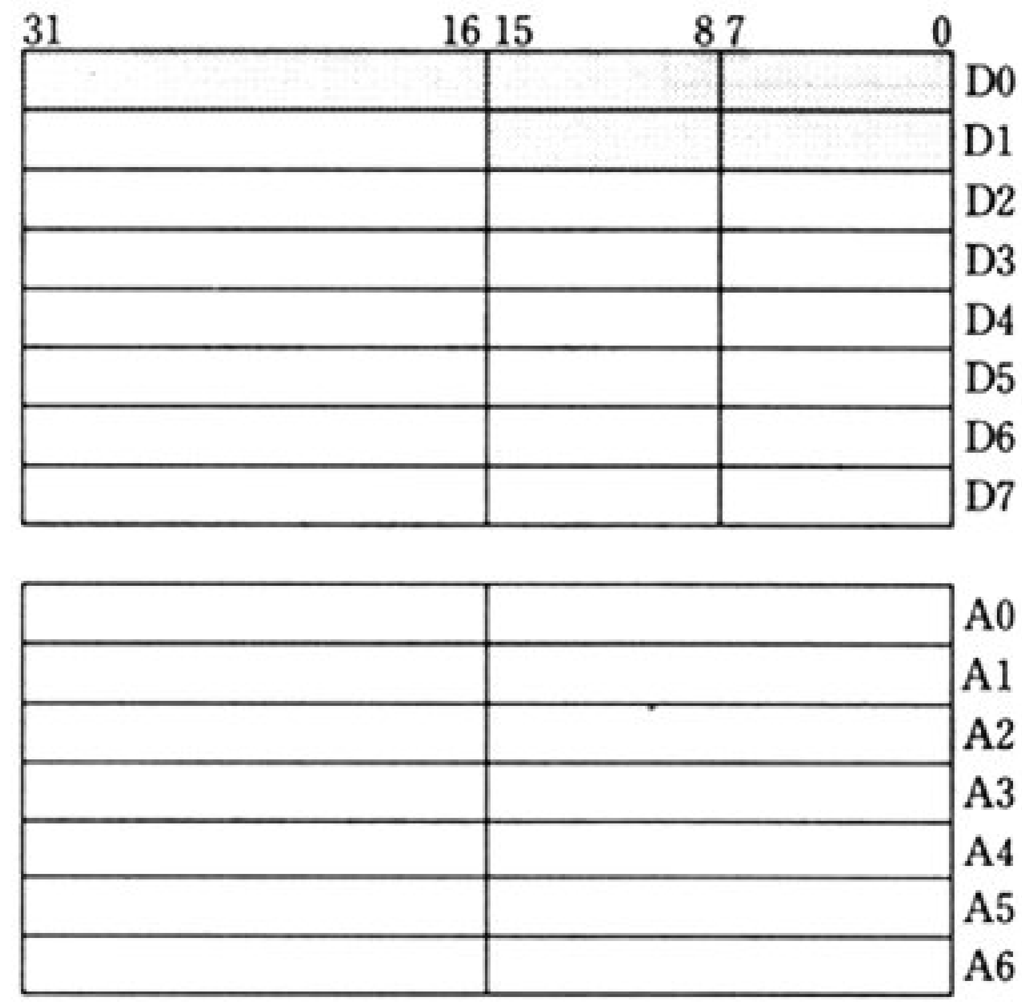
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OUTLPT,d0     * IOCSコール番号をセット
        move.w  #$0d,d1         * $0d (CR) を出力
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

OUTPRN

● OUTPRN　　IOCS コール番号：$3F



機　　能　プリンタ出力

入　　力　D0. L　　$3F　　IOCS コール番号
D1. W　　出力文字

リターン　なし（D0. L の内容は保証されません）

解　　説　プリンタ出力を行います。
このコールは、シフト JIS コードで漢字処理を行います。
入力の D1. W で出力データを指定します。
D1. W の上位バイトは 0 でなければならないので漢字は上位、下位の順で 1 バイトずつ出力してください。

プログラム例

```
*******************************************
* OUTPRN                                $3F
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OUTPRN,d0     * IOCSコール番号をセット
        move.w  #$0088,d1               * 漢字コード1byteめ
        trap    #iocs           * IOCSコール
        moveq.l #_OUTPRN,d0     * IOCSコール番号をセット
        move.w  #$00a4,d1               * 漢字コード2byteめ
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

■以降の IOCS コール番号$40〜$4F はディスク関係のコールです。
メディアと各コールとの対応表を示します。

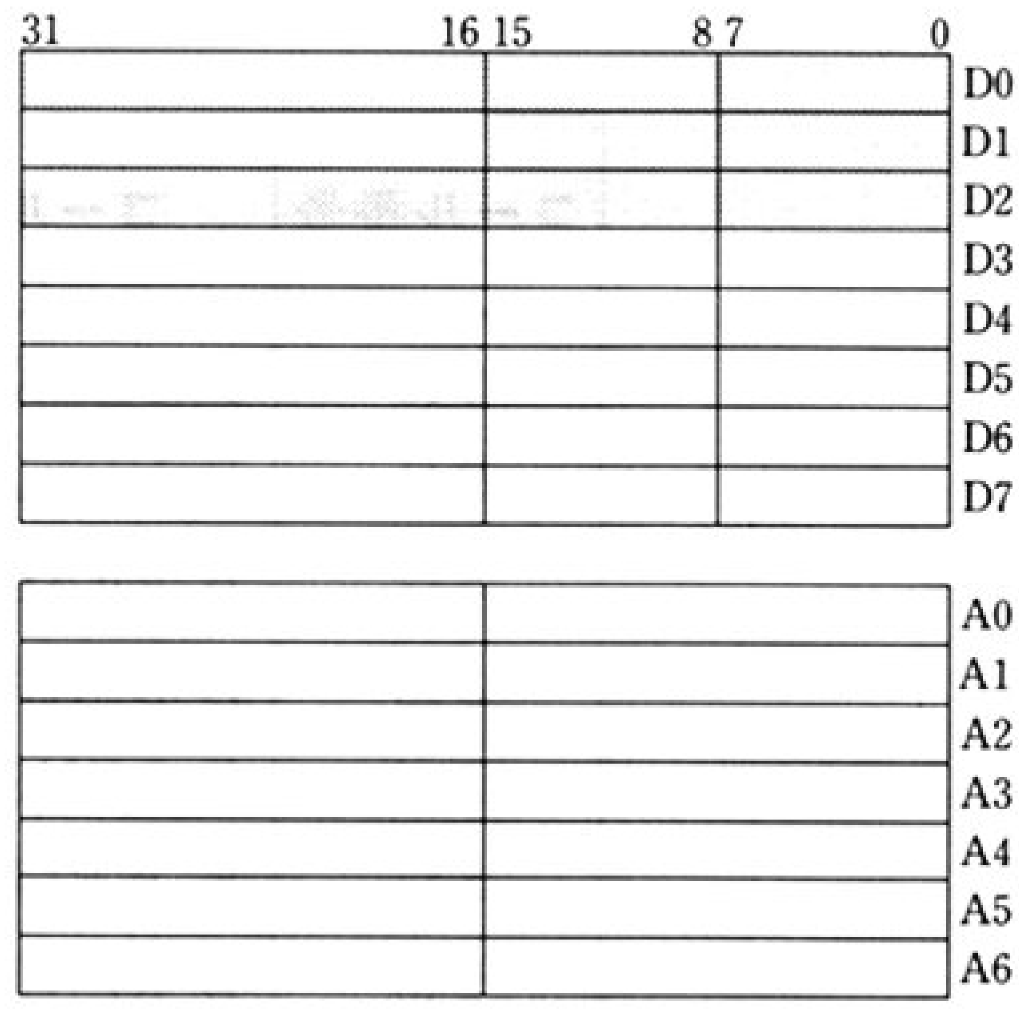
コール番号	コール名	2HD	*ハードディスク	ページ
$40	B_SEEK	○	○	334, 337
$41	B_VERIFY	○	○	339, 341
$42	B_READDI	○	×	343
$43	B_DSKINI	○	○	345, 347
$44	B_DRVSNS	◎	◎	348
$45	B_WRITE	◎	◎	349
$46	B_READ	◎	◎	351
$47	B_RECALI	○	○	353, 355
$49	B_WRITED	○	×	356
$4B	B_BADFMT	×	○	357
$4C	B_READDL	○	×	358
$4D	B_FORMAT	○	○	360, 362
$4E	B_DRVCHK	○	×	363
$4F	B_EJECT	○	○	365, 366

※この IOCS は SASI 用のハードディスク専用で、SCSI 用ハードディスクでは使用できません。

この表で、○はそのコールがそのメディアで使用できることを、×はそのコールがそのメディアで使用できないことを表しています。
どのメディアにも○がついているものは、コールの目的が同じでも内容が大きく異なるため、解説がメディアごとに分かれています。
◎は、どのメディアにとっても目的、内容ともほぼ同じであるようなコールです。
メディアによる差はそのコールの解説で述べています。

B_SEEK

● B_SEEK　　IOCS コール番号：$40 (2HD)



機　能	ディスクシーク(2HD)
入　力	D0. L　$40　IOCS コール番号 D1. W　PDA×256＋MODE D2. L　目的シーク位置
リターン	D0. L　ステータス
解　説	ディスクの指定位置までシークします。

入力の D1. W は上位 8 ビットが PDA、下位 8 ビットが MODE です。
PDA は 2HD ディスク関係のコールでは次のようになっています。

PDA＝ $90……2HD DRIVE 0
　　　⋮　　　　⋮
PDA＝ $93……2HD DRIVE 3

MODE の内容は次の通りです（他のビットはつねに 0 です）。

ビット 6：MFM …FM／MFM モード切り替え
　0 ………FM（単密度）
　1 ………MFM（倍密度）
ビット 5：RETRY
　0 ………リトライしない
　1 ………リトライする
ビット 4：SEEK
　0 ………シーク動作を行わずに実行
　1 ………シーク動作を行ってから実行

注：MFM は、システムでは MFM モードで使用します。
　　リトライする場合、10回リトライし、最初5回はシークせず、以降5回はリキャリブレート、シークをともなうリトライを行います。

入力の D2. L は、目的のシーク位置を指定します。
内容は次の通りです。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D2. L	セクタ長								トラック							

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
	サイド								セクタ							

セクタ長の指定は、次の通りです。
　0……… 128バイト
　1……… 256バイト
　2……… 512バイト
　3………1024バイト
リターンの D0. L は、FDC のステータスです。
ステータスの内容は次のとおりです。

ビット 31～24：ST0……リザルトステータス0
ビット 23～16：PCN……コマンド終了時のシリンダ番号
ビット 15～0　：不定

リザルトステータス0（ST0）の詳細については、付録6. FDC のステータスを参照してください。

プログラム例

```
*******************************************
* B_SEEK                               $40
**** 2HD **********************************

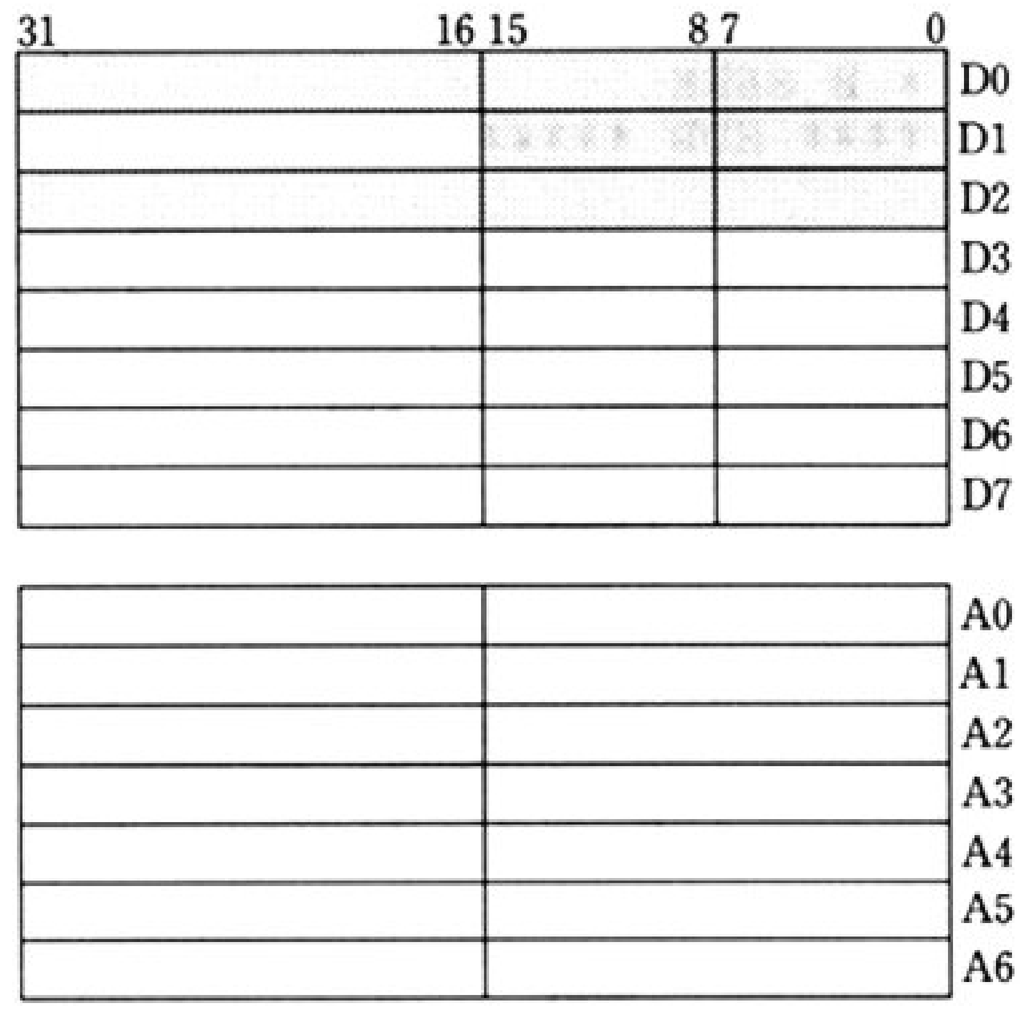
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_SEEK,d0      * IOCSコール番号をセット
        move.w  #$90_70,d1       * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

B_SEEK

● B_SEEK　　IOCS コール番号：$40（ハードディスク）



機　能　ディスクシーク（ハードディスク）

入　力
D0. L　$40　IOCS コール番号
D1. W　PDA×256＋MODE
D2. L　目的シーク位置

リターン
D0. L　ステータス
D0. L ≧ 0……………………正常終了
D0. L ＜ 0……………………異常終了
D0. L ＝ $FFFFFF??……$??がエラーコード

解　説
ディスクの指定位置までシークします。
入力の D1. W の上位 8 ビットが PDA、下位 8 ビットが MODE です。
ハードディスクの場合、MODE は意味を持ちません。
PDA はハードディスク関係のコールでは次のようになっています。

PDA＝ $80……HARD DISK DRIVE0
　　　　⋮　　　⋮
PDA＝ $8F……HARD DISK DRIVE15

入力の D2. L は、目的のシーク位置を指定します。
これは 256 バイト単位のレコード番号となります。

プログラム例

```
*******************************************
* B_SEEK                                $40
**** HDD **********************************

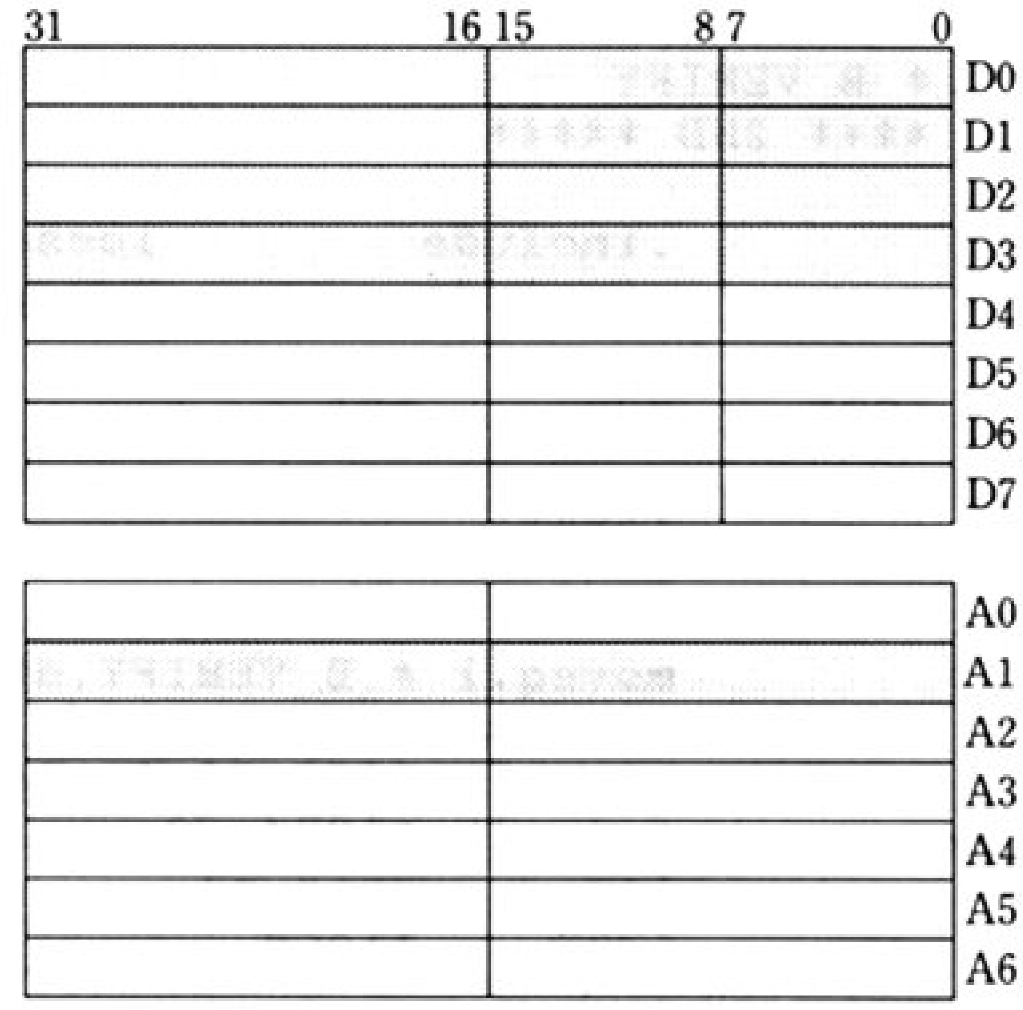
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_SEEK,d0     * IOCSコール番号をセット
        move.w  #$80_00,d1      * pda = $80
        moveq.l #1,d2           * レコード番号
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_VERIFY

● B_VERIFY　　IOCSコール番号：$41 (2HD)



機　能　データの比較チェック (2HD)

入　力

D0. L　$41　IOCSコール番号
D1. W　PDA×256+MODE
D2. L　目的チェック位置
D3. L　チェックするバイト数
A1. L　チェックするデータの先頭アドレス

リターン

D0. L　ステータス
ビット 31～24 ：ST0……リザルトステータス 0
ビット 23～16 ：ST1……リザルトステータス 1
ビット 15～ 8 ：ST2……リザルトステータス 2
ビット 7～ 0 ：　C……シリンダ番号

解　説

データの比較チェックを行います。
スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
入力のD1. W、D2. Lの詳細については、B_SEEK ($40；2HD)を参照してください。
入力のD3. L、A1. Lは、それぞれ比較チェックするデータのバイト数、先頭アドレスです。
リターンのD0. Lは、FDCのステータスです。
リザルトステータス (ST0～ST2) の詳細については、付録6. FDCのステータスを参照してください。

プログラム例

```
******************************************
* B_VERIFY                             $41
**** 2HD *********************************

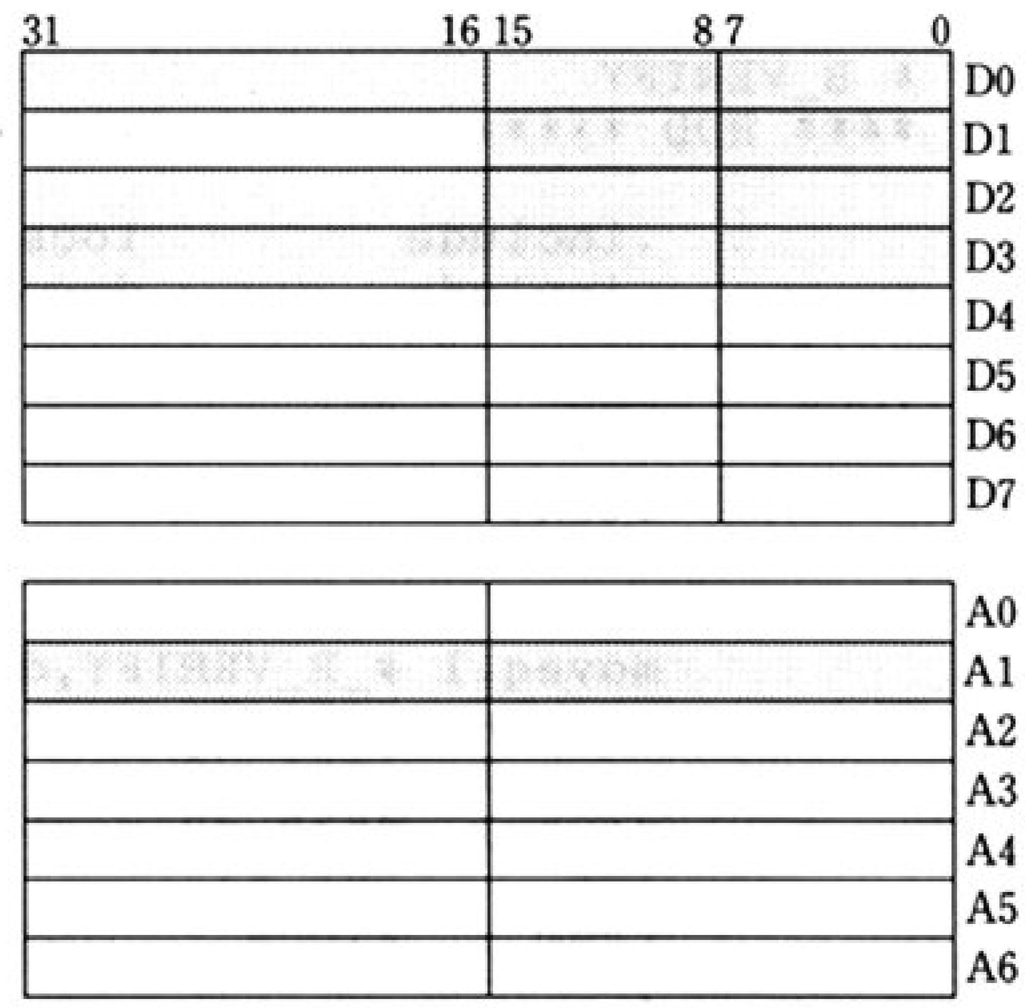
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_VERIFY,d0   * IOCSコール番号をセット
        move.w  #$90_70,d1      * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
        move.l  #1024,d3                * チェックするバイト数
        lea     data,a1         * チェックするデータの先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
data:
        dc.b    .........
        .
        .
        .end
```

B_VERIFY

● B_VERIFY　　IOCSコール番号：$41（ハードディスク）



機　能　データの比較チェック（ハードディスク）

入　力

D0. L	$41	IOCSコール番号
D1. W	PDA×256+MODE	
D2. L	目的チェック位置	
D3. L	チェックするバイト数	
A1. L	チェックするデータの先頭アドレス	

リターン　D0. L　ステータス

D0. L ＜ 0……負数なら異常終了

解　説　データの比較チェックを行います。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力のD1. W、D2. Lの詳細については、B_SEEK（$40；ハードディスク）を参照してください。

入力のD3. L、A1. Lは、それぞれ比較チェックするデータのバイト数、先頭アドレスです。

プログラム例

```
********************************************
* B_VERIFY                               $41
**** HDD ***********************************

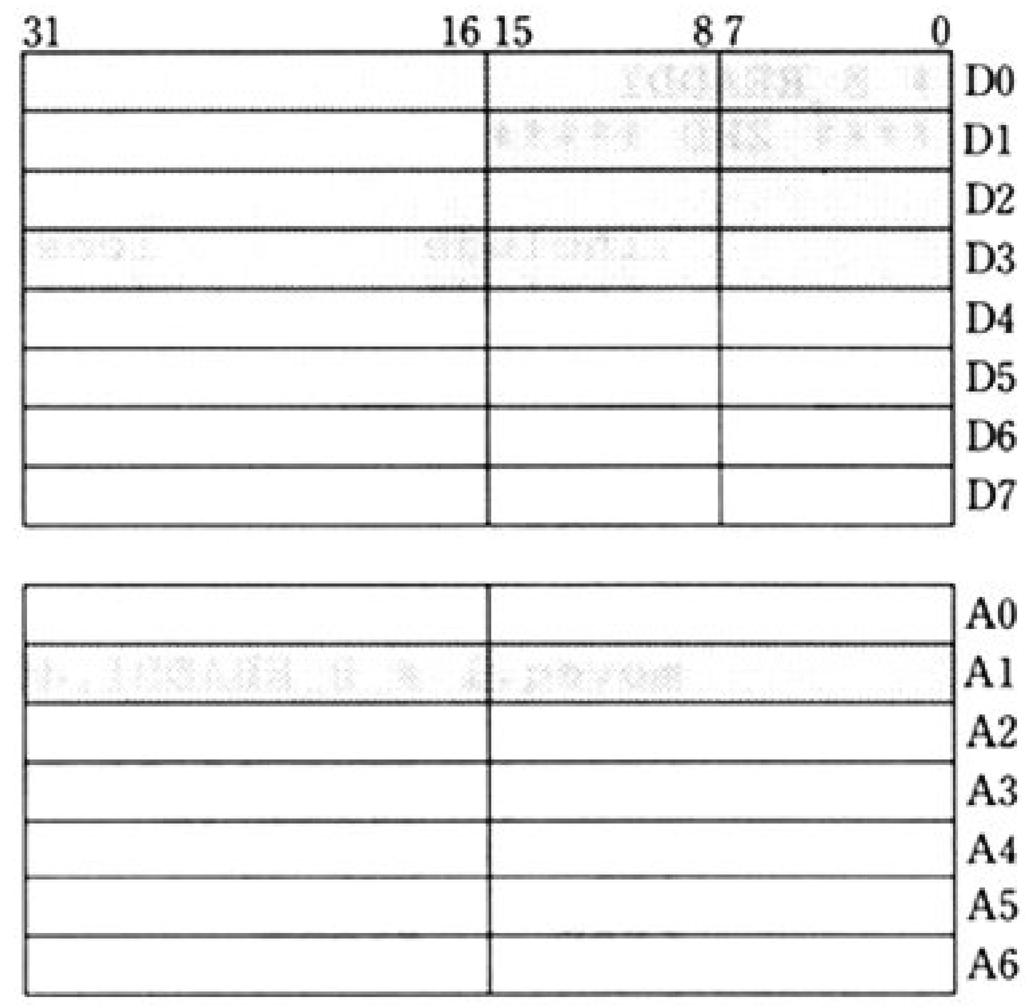
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_VERIFY,d0   * IOCSコール番号をセット
        move.w  #$80_00,d1      * pda = $80
        move.l  #$01,d2         * レコード番号
        move.l  #512,d3         * チェックするバイト数
        lea     buf,a1          * チェックするデータの先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
buf:    dc.b    …………
        .
        .
        .end
```

B_READDI

● B_READDI　　IOCSコール番号：$42



機　能　診断のための読み取り（2HDのみ）

入　力

D0. L	$42　IOCSコール番号
D1. W	PDA×256＋MODE
D2. L	目的読み取り位置
D3. L	読み込むバイト数
A1. L	読み込み領域の頭アドレス

リターン　D0. L　ステータス

ビット 31〜24：ST0……リザルトステータス 0
ビット 23〜16：ST1……リザルトステータス 1
ビット 15〜 8：ST2……リザルトステータス 2
ビット 7〜 0：　C……シリンダ番号

解　説　診断のための読み取り（2HDのみ）動作を行います。

スーパーバイザー領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力のD1. W、D2. Lの詳細については、B_SEEK（$40；2HD）を参照してください。

入力のD3. L、A1. Lは、それぞれ読み込むデータのバイト数、先頭アドレスです。

リターンのD0. Lは、FDCのステータスです。

リザルトステータス（ST0〜ST2）の詳細については、付録6. FDCのステータスを参照してください。

プログラム例

```
*******************************************
* B_READDI                                $42
**** 2HD **********************************

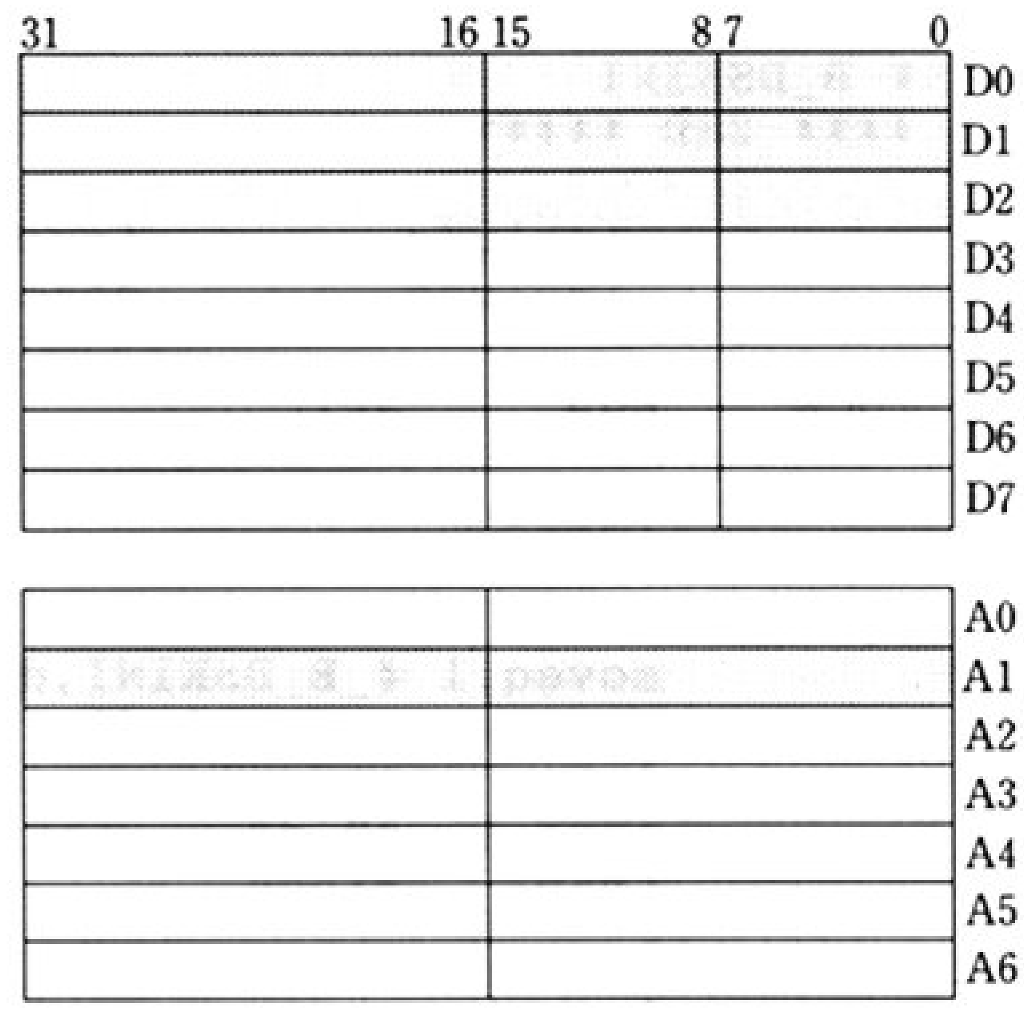
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_READDI,d0   * IOCSコール番号をセット
        move.w  #$90_70,d1      * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
        move.l  #1024,d3                * 読み取りバイト数
        lea     buf,a1          * 読み込みバッファ先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
buf:    ds.b 1024
        .
        .
        end
```

B_DSKINI

● B_DSKINI　　IOCS コール番号：$43 (2HD)



機　　能　ディスクの初期化(2HD)

入　　力

D0. L	$43	IOCS コール番号
D1. W	PDA×256(下位バイトは0)	
A1. L	SPECIFYコマンドのデータアドレス	
D2. L	モーターオフまでの時間 (1/100 秒)	

リターン　D0. L　ステータス
ビット 31～24：ST3……リザルトステータス 3
ビット 23～ 0：不定

解　　説

ディスクの初期設定を行います。

入力の D1. W ($9?00)の詳細については、B_SEEK($40；2HD)を参照してください。

入力の A1. L が 0 のときはデフォルト値を、それ以外のときは SPECIFY コマンドのデータアドレスを指定します。

入力の D2. W が 0 のときはデフォルト値（2 秒）を、それ以外のときはモーターオフまでの時間（0.01 秒単位）を指定します。

リターンの D0. L は、FDC のステータスです。

リザルトステータス（ST0～ST2）の詳細については、付録 6. FDC のステータスを参照してください。

プログラム例

```
*******************************************
* B_DSKINI                              $43
**** 2HD **********************************

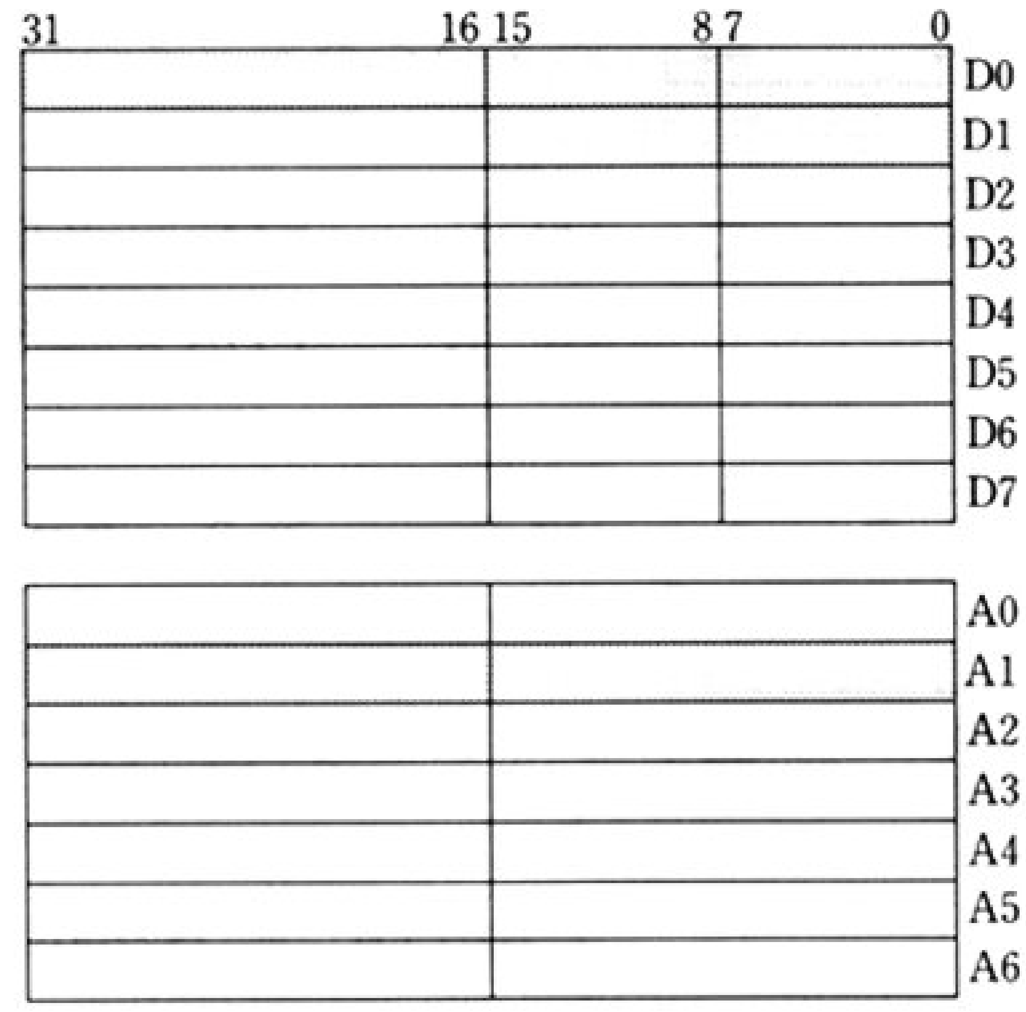
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_DSKINI,d0   * IOCSコール番号をセット
        move.w  #$90_00,d1      * pda = $90
        lea     cmd,a1          * SPECIFY コマンドデータ先頭アドレス
        move.l  #0,d2           * デフォルトで 2 秒
        trap    #iocs           * IOCSコール
        .
        .
        .data
cmd:    dc.w    03,00,00                *SPECIFYコマンド
        .
        .
        .end
```

B_DSKINI

● B_DSKINI　　IOCSコール番号：$43（ハードディスク）



機　能　ディスクの初期化（ハードディスク）

入　力
D0. L　$43　IOCSコール番号
D1. W　PDA×256（下位バイトは0）
A1. L　アサインドライブパラメータのデータアドレス

リターン　D0. L　ステータス
D0. L ＜ 0……異常終了

解　説　ディスクの初期設定を行います。
入力のD1. W ($8?00) の詳細については、B_SEEK（$40；ハードディスク）を参照してください。
入力のA1. Lが0のときはデフォルト値を、それ以外のときはアサインドライブパラメータのデータアドレスを指定します。

プログラム例

```
********************************************
* B_DSKINI                              $43
**** HDD ***********************************

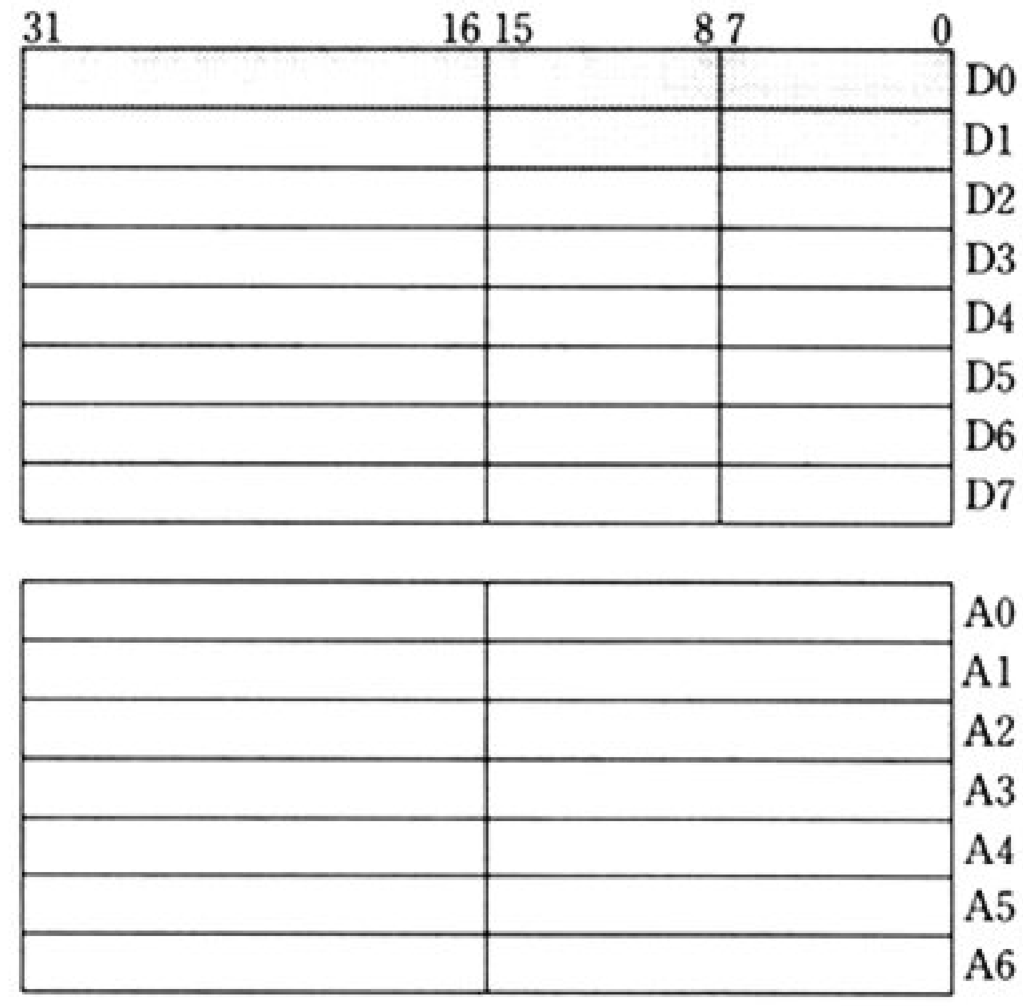
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_DSKINI,d0    * IOCSコール番号をセット
        move.w  #$80_00,d1       * pda = $80
        lea     prm,a1           * アサインドライブパラメータのアドレス
        trap    #iocs            * IOCSコール
        .
        .
        .data
prm:
        .
        .
        .end
```

B_DRVSNS

● B_DRVSNS　　　IOCS コール番号：$44



機　能　ディスクステータスのチェック

入　力　D0. L　$44　IOCS コール番号
D1. W　PDA×256(下位バイトは0)

リターン　D0. L　ステータス

解　説　ディスクのステータスをチェックします。
入力の D1. W ($??00) の PDA については、B_SEEK ($40；2HD、ハードディスク) を参照してください。
リターンの D0. L ステータスについては、B_DSKINI ($43；2HD、ハードディスク) を参照してください。

プログラム例

```
*******************************************
* B_DRVSNS                             $44
*******************************************

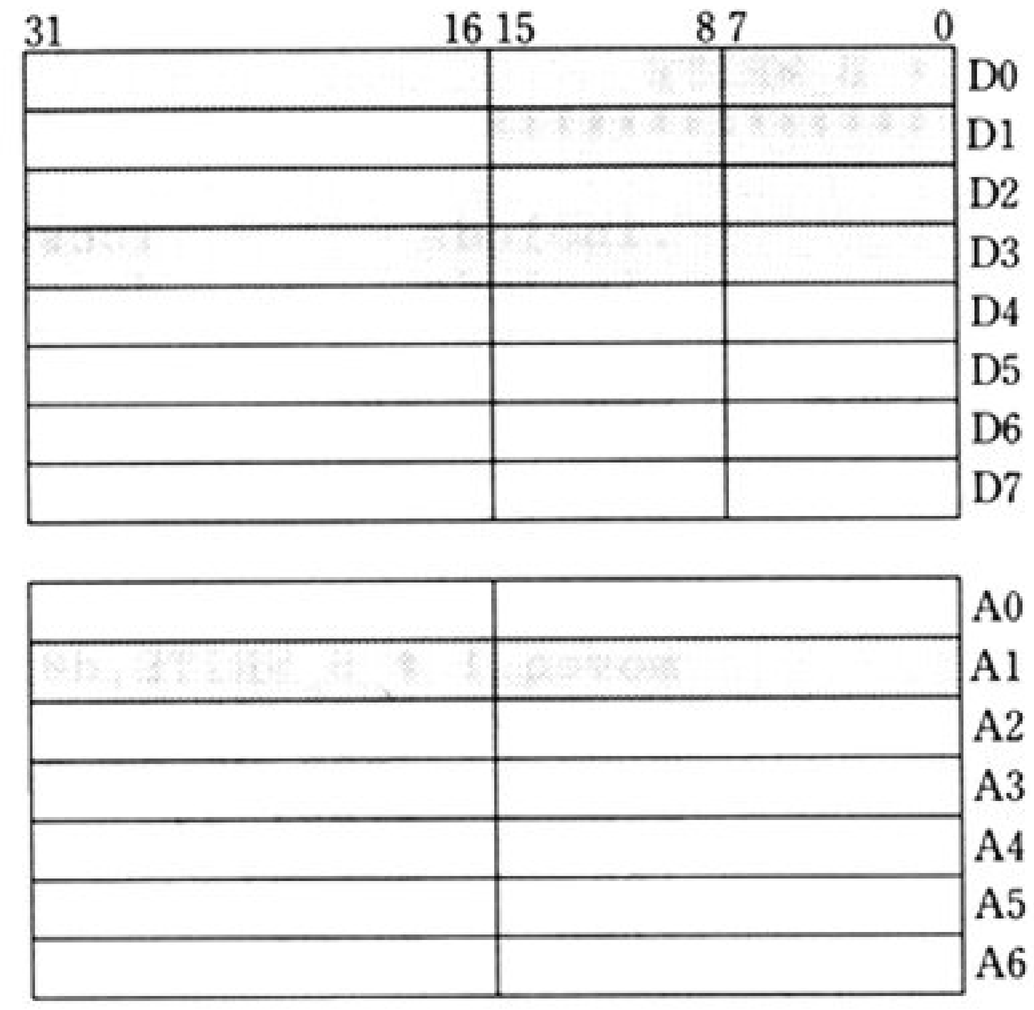
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_DRVSNS,d0    * IOCSコール番号をセット
        move.w  #$80_00,d1       * pda = $80
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

B_WRITE

● B_WRITE　　　IOCS コール番号：$45



機　　能	ディスクデータ書き込み	
入　　力	D0. L	$45　IOCS コール番号
	D1. W	PDA×256＋MODE
	D2. L	目的書き込み位置
	D3. L	書き込みバイト数
	A1. L	書き込みデータの先頭アドレス
リターン	D0. L	ステータス

解　　説

ディスクにデータを書き込みます。

スーパーバイザ領域のアドレスも指定できますが、不正なアドレスを指定しないように注意してください。

入力の D1. W、D2. L については、B_SEEK（$40；2HD、ハードディスク）を参照してください。

入力の D3. L、A1. L は、それぞれ書き込むデータのバイト数、先頭アドレスです。

リターンの D0. L のステータスについては、B_VERIFY（$41；2HD、ハードディスク）を参照してください。

注：OS が行っているバッファリング処理とは無関係に動作しますので、OS がバッファ中にデータを保持している場合、ディスク上のデータは内容が保証されない場合があります。

プログラム例

```
********************************************
* B_WRITE                               $45
********************************************

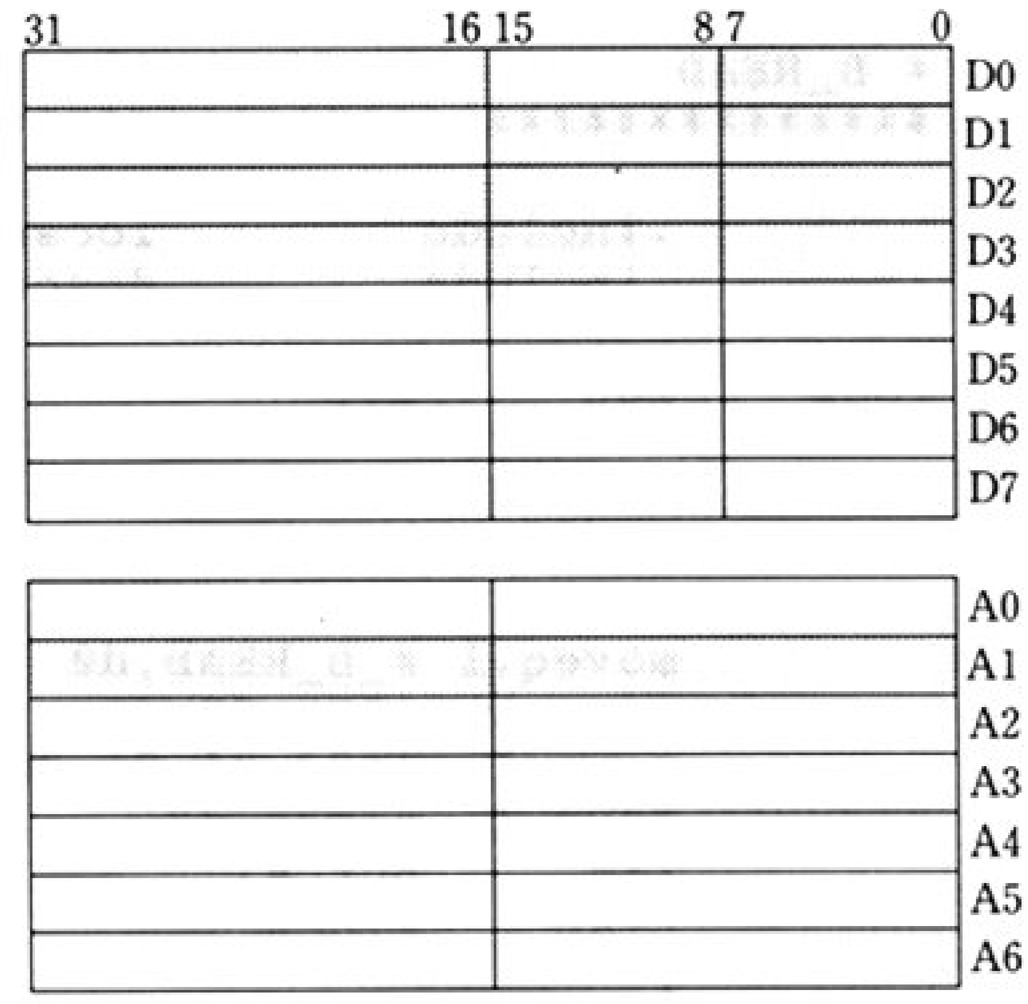
        .include        iocscall.mac
        .include        doscall.mac

iocs    equu    $0f

        .text
        .
        .
        moveq.l #_B_WRITE,d0    * IOCSコール番号をセット
        move.w  #$90_70,d1      * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
        move.l  #1024,d3        * 書き込みバイト数
        lea     buf,a1          * 書き込みデータ先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
buf:    dc.b    .........
        .
        .
        .end
```

B_READ

● B_READ　　IOCS コール番号：$46



機　能　ディスクデータ読み込み

入　力

D0. L	$46	IOCS コール番号
D1. W	PDA×256+MODE	
D2. L	目的読み込み位置	
D3. L	読み込みバイト数	
A1. L	読み込み領域の先頭アドレス	

リターン　D0. L　ステータス

解　説

ディスクからデータを読み込みます。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力の D1. W、D2. L については、B_SEEK（$40；2HD、ハードディスク）を参照してください。

入力の D3. L、A1. L は、それぞれ読み込むデータのバイト数、先頭アドレスです。

リターンの D0. L のステータスについては、B_VERIFY（$41；2HD、ハードディスク）を参照してください。

注：OS が行っているバッファリング処理とは無関係に動作しますので、OS がバッファ中にデータを保持している場合、ディスク上のデータは内容が保証されない場合があります。

プログラム例

```
*******************************************
* B_READ                              $46
*******************************************

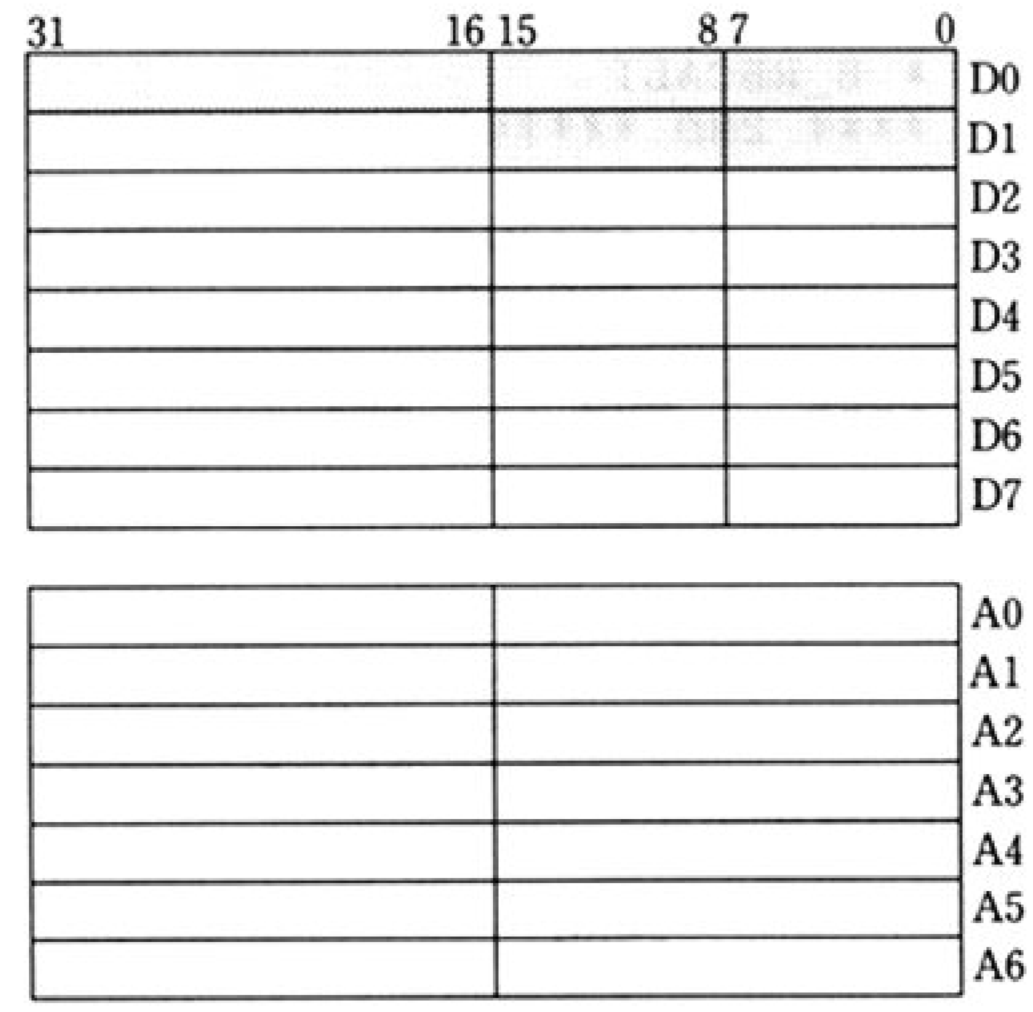
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_READ,d0      * IOCSコール番号をセット
        move.w  #$90_70,d1       * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
        move.l  #1024,d3         * 読み取りバイト数
        lea     buf,a1           * 読み込みバッファ先頭アドレス
        trap    #iocs            * IOCSコール
        .
        .
        .data
buf:    ds.b 1024
        .
        .
        .end
```

B_RECALI

● B_RECALI　　　IOCSコール番号：$47 (2HD)



機　能　トラック0へのシーク(2HD)

入　力　D0. L　$47　IOCSコール番号
D1. W　PDA×256＋MODE

リターン　D0. L　ステータス

解　説　トラック0へのシーク、または強制レディ状態でのチェックを行います。
入力のD1. Wは上位8ビットがPDA、下位8ビットがMODEです。
2HDディスク関係のコールでのPDAは、次のようになっています。

PDA ＝ $90……2HD DRIVE0
　　　　⋮　　　　⋮
PDA ＝ $93……2HD DRIVE3

MODEは0か$FFを指定します。

MODE ＝ 0……トラック0へのシーク
MODE ＝ $FF……強制レディ状態でのチェック

2HDのドライブの強制レディ状態でのチェックを行ったときに、このコール実行後にリザルトステータス0のビット4 (EQUIPMENT CHECK)が1になっていた場合、そのドライブは存在しないことを示します。
2HDドライブの各ドライブの有無は、このコールでしかチェックできません。
リターンのD0. Lのステータスについては、B_SEEK($40；2HD)を参照してください。

プログラム例

```
*******************************************
* B_RECALI                            $47
**** 2HD **********************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equu    $0f

        .text
        .
        .
        moveq.l #_B_RECALI,d0   * IOCSコール番号をセット
        move.w  #$90_00,d1      * pda = $90
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_RECALI

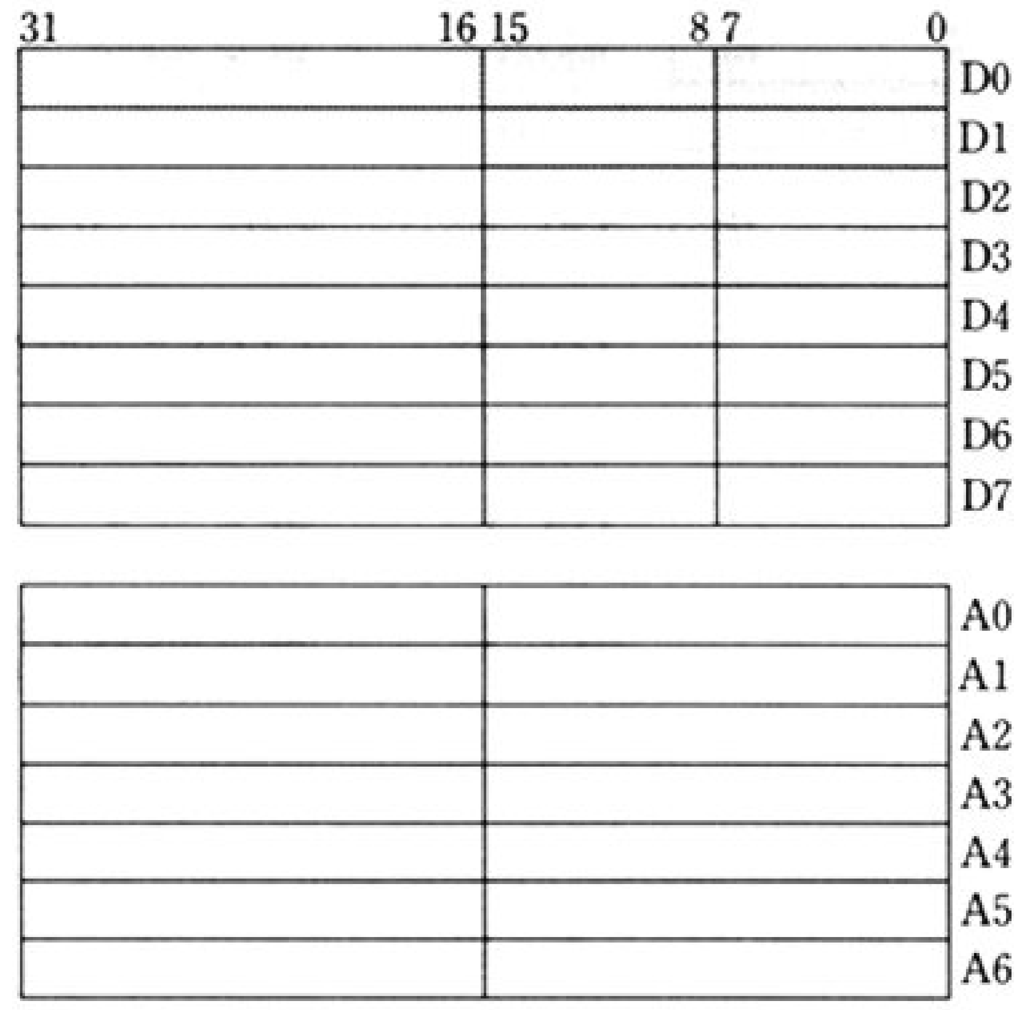
● B_RECALI　　IOCSコール番号：$47（ハードディスク）

機　能　トラック0へのシーク(ハードディスク)

入　力　D0. L　$47　IOCSコール番号
D1. W　PDA×256(下位バイトは0)

リターン　D0. L　ステータス

解　説　トラック0へのシークを行います。
入力のD1. W、リターンのD0. Lのステータスについては、B_SEEK（$40；ハードディスク）を参照してください。



プログラム例

```
*********************************************
* B_RECALI                               $47
**** HDD ************************************

        .include        iocscall.mac
        .include        doscall.mac

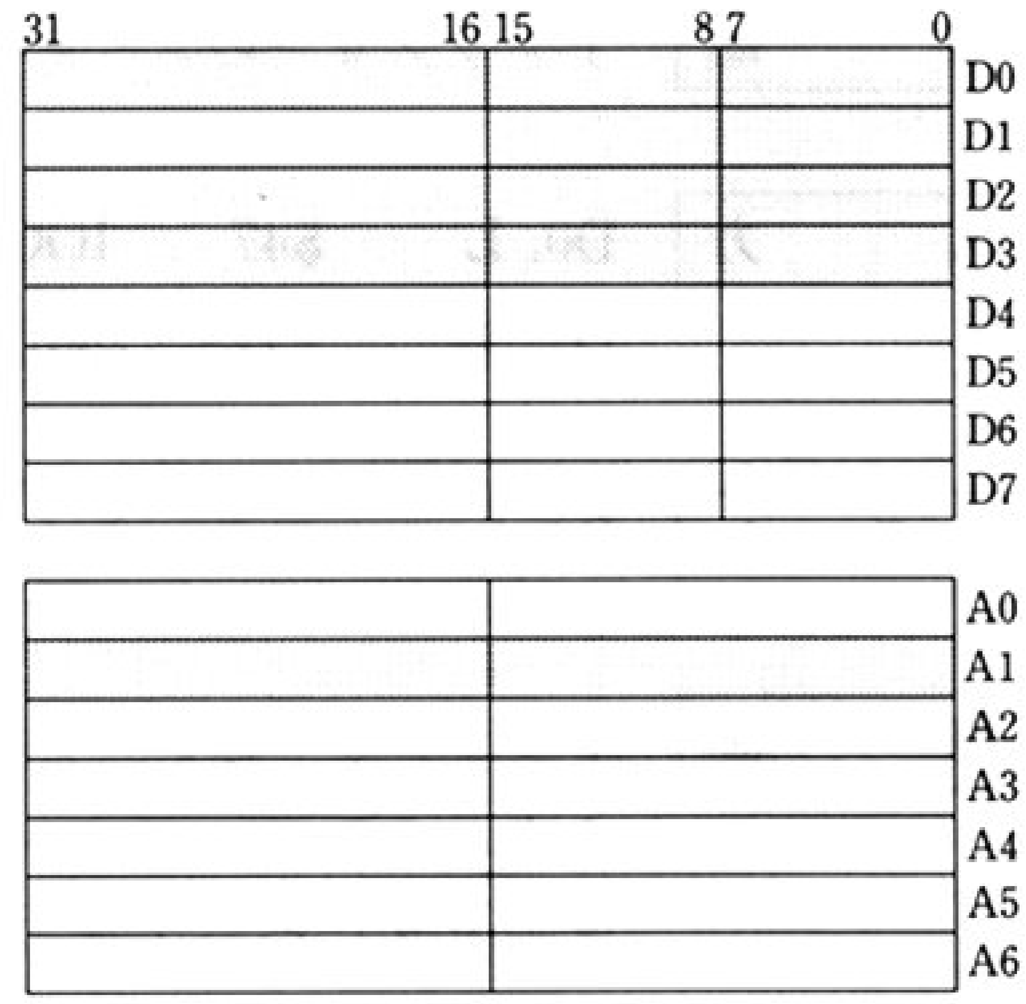
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_RECALI,d0    * IOCSコール番号をセット
        move.w  #$80_00,d1       * pda = $80
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

B_WRITED

● B_WRITED　　IOCS コール番号：$49

機　能	破損データ書き込み（2HD のみ）	
入　力	D0. L	$49　IOCS コール番号
	D1. W	PDA×256＋MODE
	D2. L	目的書き込み位置
	D3. L	書き込みバイト数
	A1. L	書き込みデータの先頭アドレス



リターン　D0. L　ステータス

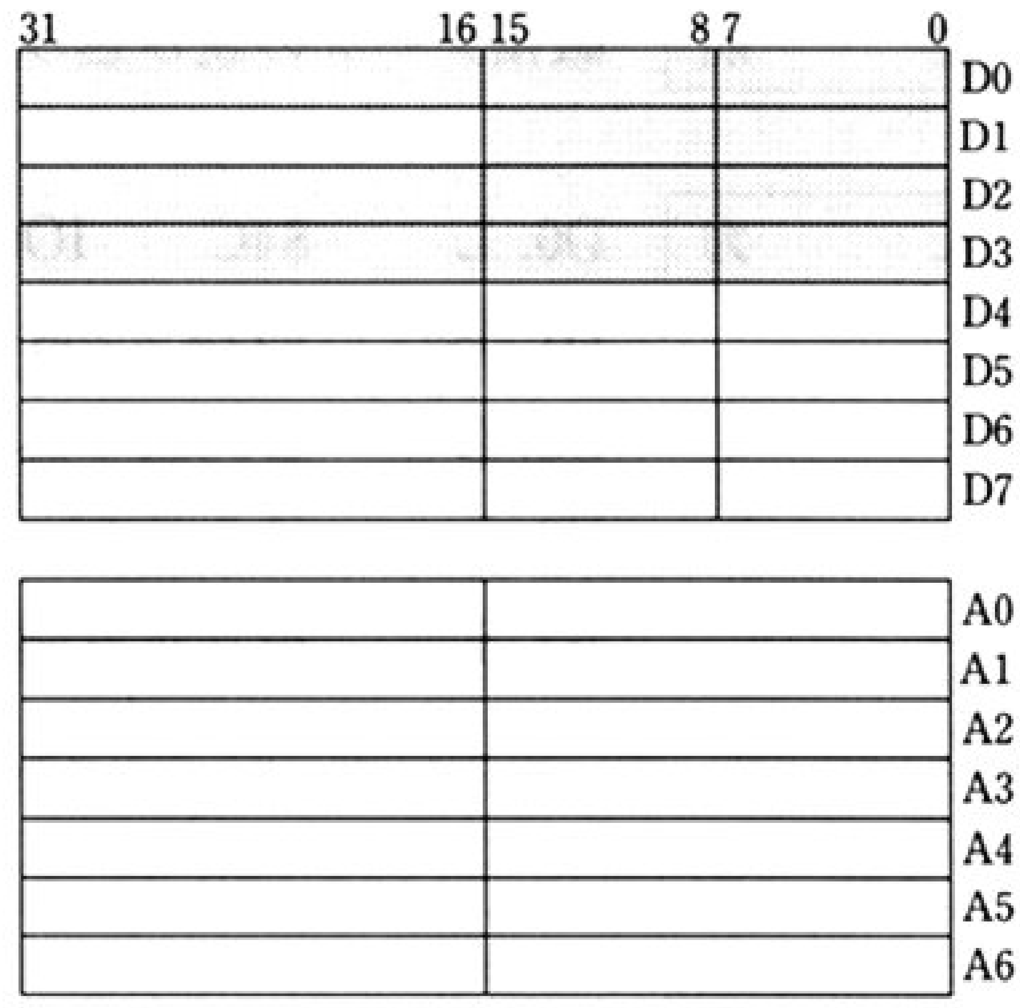
ビット 31～24：ST 0……リザルトステータス 0
ビット 23～16：ST 1……リザルトステータス 1
ビット 15～ 8：ST 2……リザルトステータス 2
ビット 7～ 0：　C……シリンダ番号

解　説

2HD ディスクへの破損データの書き込みを行います。
スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
入力の D1. W、D2. L の詳細については、B_SEEK($40；2HD)を参照してください。
入力の D3. L、A1. L はそれぞれ書き込むデータのバイト数、先頭アドレスです。
リターンの D0. L は、FDC のステータスです。
リザルトステータス（ST0～ST2）の詳細については、付録 6. FDC のステータスを参照してください。

B_BADFMT

● B_BADFMT　　IOCS コール番号：$4B



機　能　破壊トラックを使用不能にする
(ハードディスクのみ)

入　力
D0. L　$4B　IOCS コール番号
D1. W　PDA×256（下位バイトは0）
D2. L　レコード番号
D3. L　インターリーブコード（1または6）

リターン　D0. L　終了コード
D0. L ≦ 0……正常終了
D0. L ＞ 0……異常終了

解　説　ハードディスクの破壊トラックを使用不能にします。
入力のD1. W（PDA×256）の内容は次の通りです。

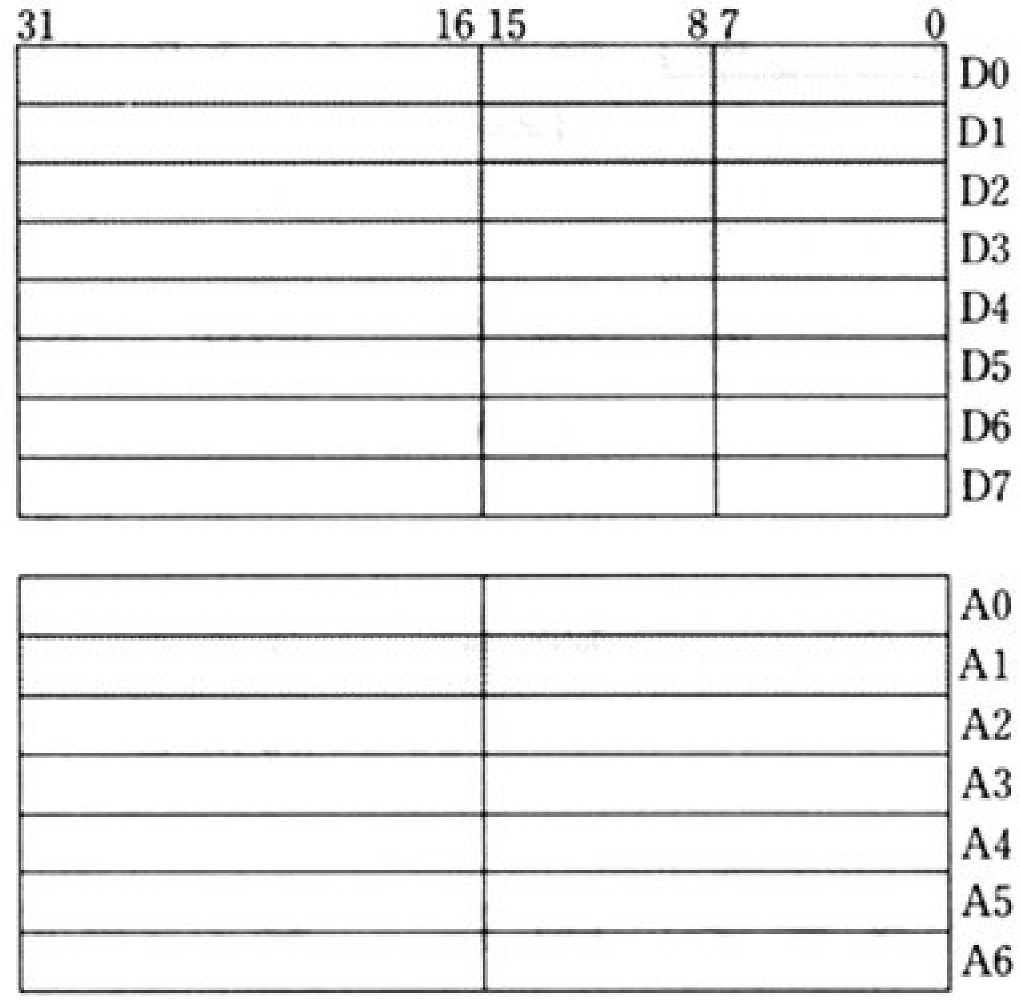
D1. W ＝ $8000……HARD DISK DRIVE0
⋮　⋮
D1. W＝ $8F00……HARD DISK DRIVE15

入力のD2. Lは、レコード番号（256バイト単位）です。
入力のD3. Lは、インターリーブコードで1または6を指定します。

B_READDL

● B_READDL　　IOCS コール番号：$4C

機　能　破損データの読み込み（2HD のみ）



入　力

D0. L	$4C	IOCS コール番号
D1. W	PDA×256＋MODE	
D2. L	目的読み込み位置	
D3. L	読み込みバイト数	
A1. L	読み込み領域の先頭アドレス	

リターン　D0. L　ステータス

ビット 31～24：ST 0……リザルトステータス 0
23～16：ST 1……リザルトステータス 1
15～ 8：ST 2……リザルトステータス 2
7～ 0：　C……シリンダ番号

解　説　2HD ディスクから破損データを読み込みます。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力の D1. W、D2. L の詳細については、B_SEEP($40；2HD)を参照してください。

入力の D3. L、A1. L は、それぞれ読み込むデータのバイト数、先頭アドレスです。

リターンの D0. L は、FDC のステータスです。

リザルトステータス（ST0～ST2）の詳細については、付録 6. FDC のステータスを参照してください。

プログラム例

```
****************************************
* B_READDL                          $4C
****************************************

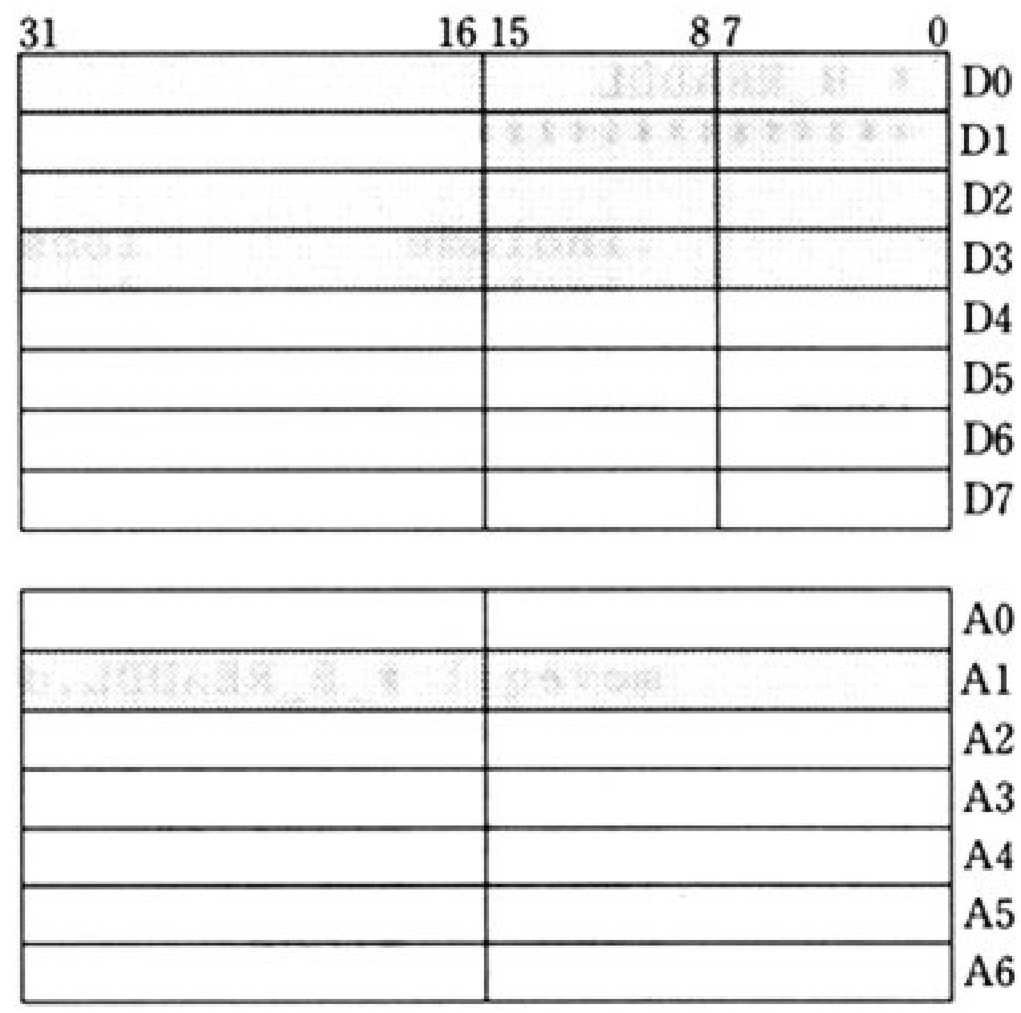
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_READDL,d0    * IOCSコール番号をセット
        move.w  #$90_70,d1       * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_08,d2* sector長=3,track=5,side=1,sector=8
]       move.l  #1024,d3         * 読み込みバイト数
        lea     buf,a1           * 読み込みバッファ先頭アドレス
        trap    #iocs            * IOCSコール
        .
        .
        .data
buf:    ds.b 1024
        .
        .
        .end
```

B_FORMAT

● B_FORMAT　　　IOCS コール番号：$4D (2HD)



機　能　物理フォーマット(2HD)

入　力

D0. L	$4D	IOCS コール番号
D1. W	PDA×256＋MODE	
D2. L	目的フォーマット位置	
D3. L	バイト数（ID データのバイト数）	
A1. L	ID データ先頭アドレス	

リターン　D0. L　　ステータス

解　説　物理フォーマットを行います。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力の D1. W の詳細については、B_SEEP($40；2HD)を参照してください。

入力の D2. L は目的のフォーマット位置と、フォーマット時に書き込むデータを指定します。

内容は次の通りです。

D2. L

D31–D24	D23–D16
セクタ長	トラック

D15–D08	D07–D00
サイド	データ

入力の D3. L、A1. L はそれぞれ ID データのバイト数、ID データ先頭アドレスです。

リターンの D0. L については、B_VERIFY($41；2HD)を参照してください。

プログラム例

```
*********************************************
* B_FORMAT                               $4D
**** 2HD ************************************

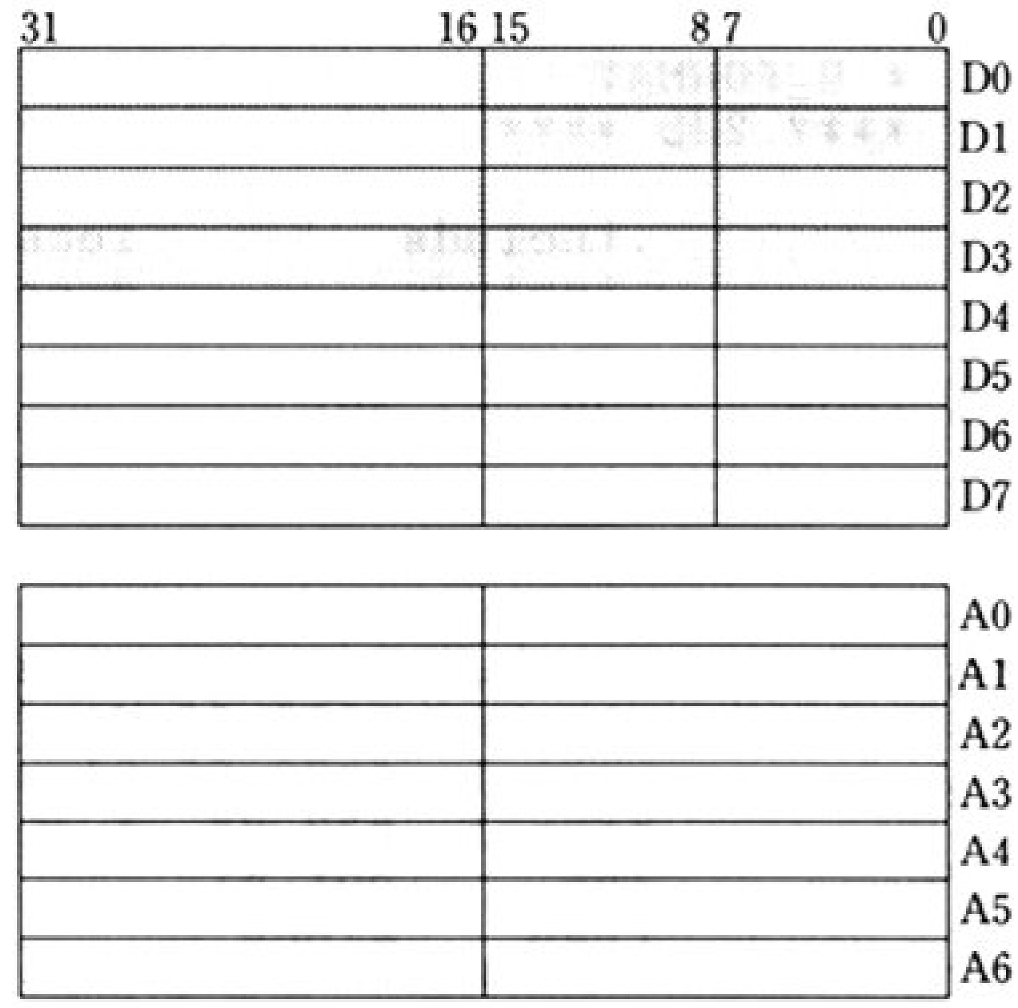
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_FORMAT,d0    * IOCSコール番号をセット
        move.w  #$90_70,d1       * pda = $90、mode = %0111_0000 = $70
        move.l  #$03_05_01_e5,d2* sector長=3,track=5,side=1,data=e5
        move.l  #32,d3  * バイト数
        lea     buf,a1           * ID データ先頭アドレス
        trap    #iocs            * IOCSコール
        .
        .
        .data
buf:
        dc.b 5,1,1,3
        dc.b 5,1,2,3
        dc.b 5,1,3,3
        dc.b 5,1,4,3
        dc.b 5,1,5,3
        dc.b 5,1,6,3
        dc.b 5,1,7,3
        dc.b 5,1,8,3
        .
        .
        .end
```

B_FORMAT

● B_FORMAT　　IOCS コール番号：$4D（ハードディスク）



機　能　物理フォーマット（ハードディスク）

入　力

D0. L	$4D	IOCS コール番号
D1. W	PDA×256（下位バイトは 0）	
D2. L	目的フォーマット位置	
D3. L	インターリーブコード（1 または 6）	

リターン　D0. L　ステータス

解　説　物理フォーマットを行います。

入力の D1. W、D2. L の詳細については、B_SEEK（$40；ハードディスク）を参照してください。

入力の D3. L はインターリーブコードで、1 または 6 を指定します。

リターンの D0. L については、B_VERIFY（$41；ハードディスク）を参照してください。

プログラム例

```
**********************************************
* B_FORMAT                                $4D
**** HDD *************************************

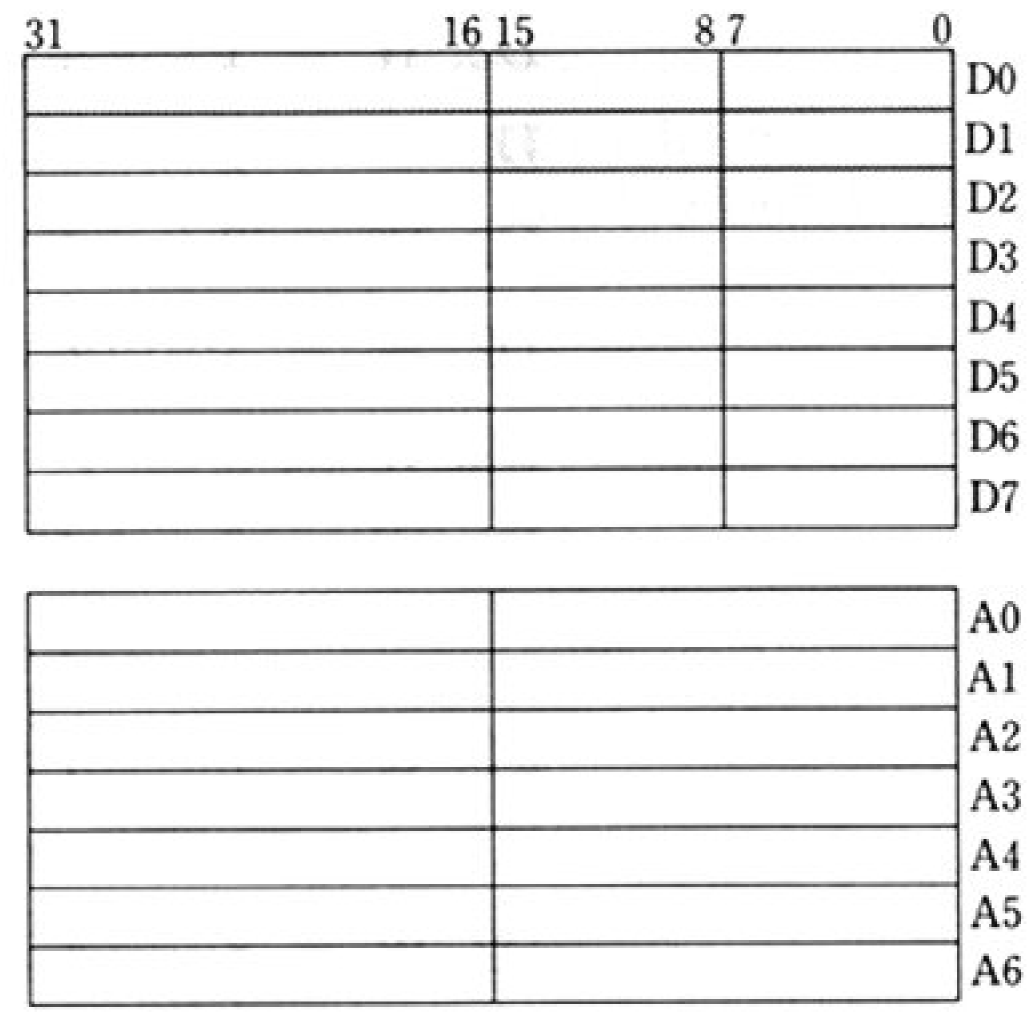
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_FORMAT,d0   * IOCSコール番号をセット
        move.w  #$80_00,d1      * pda = $80
        move.l  #$01,d2         * レコード 1
        moveq.l #1,d3           * インターリーブコード 1
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_DRVCHK

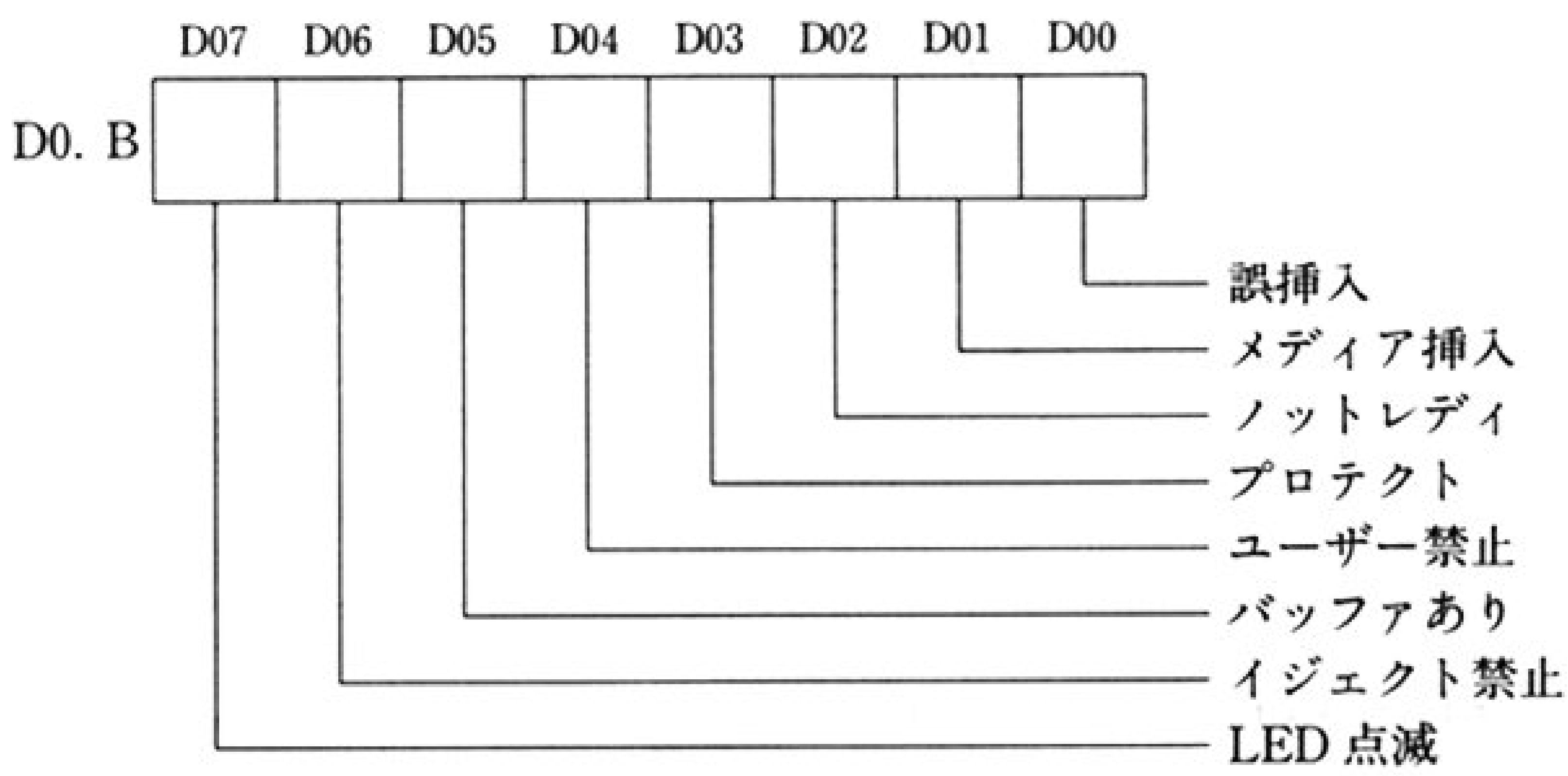
● B_DRVCHK　　IOCSコール番号：$4E



機　能　ドライブ状態の参照／設定（2HDのみ）

入　力　D0. L　$4E　IOCSコール番号
D1. W　PDA×256（下位バイトは0）
D2. W　機能番号

リターン　D2. W ＝ 0〜7, 9：
D0. B　ドライブ状態



D2. W ＝ 0のときでメディア挿入のときのみ、プロテクトとノットレディが1になる可能性があります。
その他の場合0になります。

D2. W ＝ 8のときは
D0. B ＝ 1……イジェクトされていない
D0. B ＝ −1…イジェクト状態

解　説　2HDディスクの指定ドライブの状態を参照、設定します。
入力のD1. W（PDA×256）の内容は次の通りです。

D1. W ＝ $9000…2HD DRIVE0
⋮　　⋮
D1. W ＝ $9300…2HD DRIVE3

入力の D2. W は機能番号で、内容は次の通りです。

D2. W ＝ 0……状態を検査

D2. W ＝ 1……イジェクト（イジェクト禁止状態では不可）

D2. W ＝ 2……イジェクト禁止 1（機能番号 1 のイジェクトも禁止）

D2. W ＝ 3……イジェクト許可 1

D2. W ＝ 4……ディスクがセットされていないとき、LED 点滅

D2. W ＝ 5……ディスクがセットされていないとき、LED 消灯

D2. W ＝ 6……イジェクト禁止 2（機能番号 1 のイジェクトも禁止する）
OS 用なのでユーザーは使用禁止

D2. W ＝ 7……イジェクト許可 2
OS 用なのでユーザーは使用禁止

D2. W ＝ 8……イジェクト検査
前回の検査後にイジェクトしたかどうかの検査
OS 用なのでユーザーは使用禁止

D2. W ＝ 9……状態を検査
機能番号 0 とは異なりノットレディとプロテクトは常に 0 になります

プログラム例

```
*******************************************
* B_DRVCHK                             $4E
**** 2HD **********************************

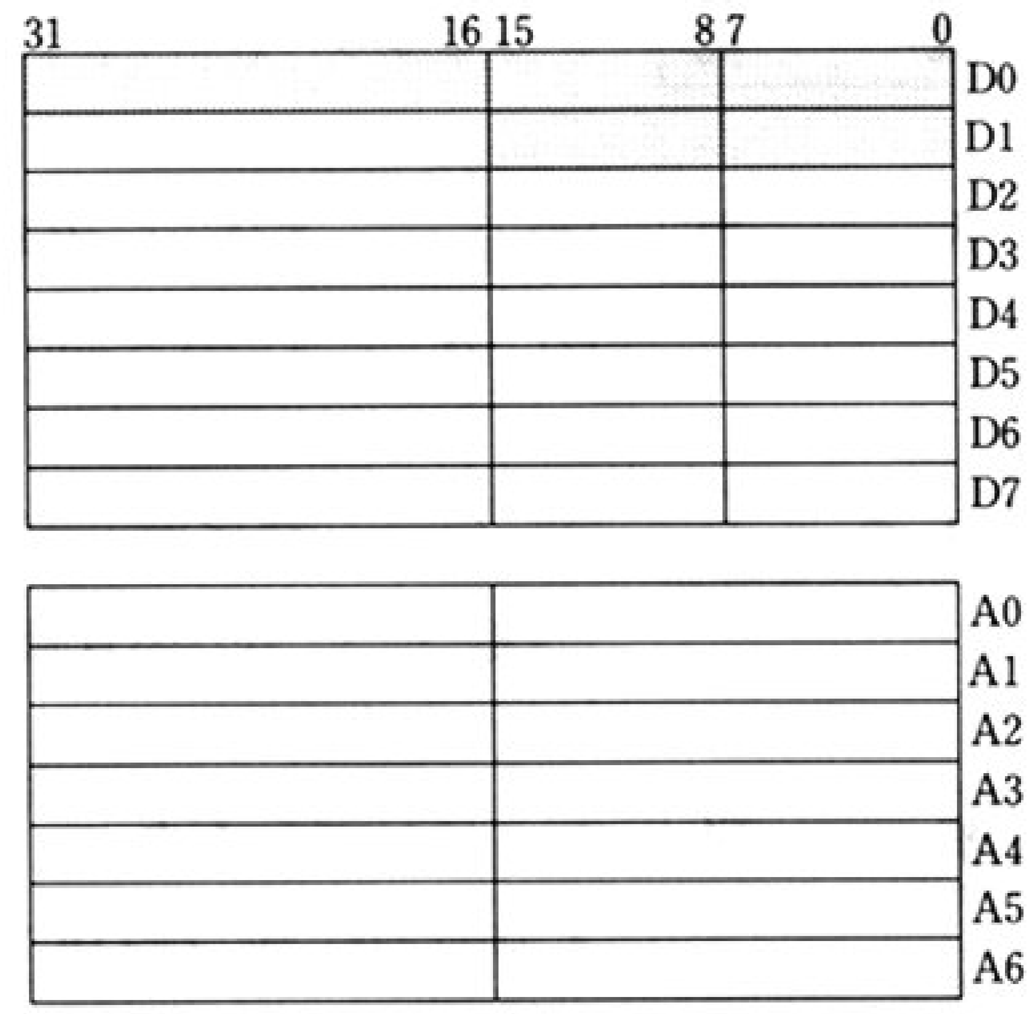
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_DRVCHK,d0   * IOCSコール番号をセット
        move.w  #$90_00,d1      * pda = $90
        move.w  #0,d2           * 状態検査
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_EJECT

● B_EJECT　　IOCSコール番号：$4F (2HD)



機　　能　イジェクト(2HD)

入　　力　D0. L　　$4F　　IOCSコール番号
・D1. W　　PDA×256(下位バイトは0)

リターン　なし (D0. Lの内容は保証されません)

解　　説　イジェクトしますが、イジェクト禁止状態でも実行可能です。
入力のD1. W (PDA×256) は、2HDディスク関係のコールでは次のようになっています。

D1. W ＝ $9000……2HD DRIVE0
　　　　　⋮　　　　　⋮
D1. W ＝ $9300……2HD DRIVE3

注：OSが行っているバッファリング処理とは無関係に動作しますので、OSがバッファ中にデータを保持している場合、イジェクトを行うとディスクの内容が保証されなくなりますので、イジェクトしてはいけません。

プログラム例

```
********************************************
* B_EJECT                                $4F
**** 2HD ***********************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_EJECT,d0    * IOCSコール番号をセット
        move.w  #$90_00,d1      * pda = $90
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

B_EJECT

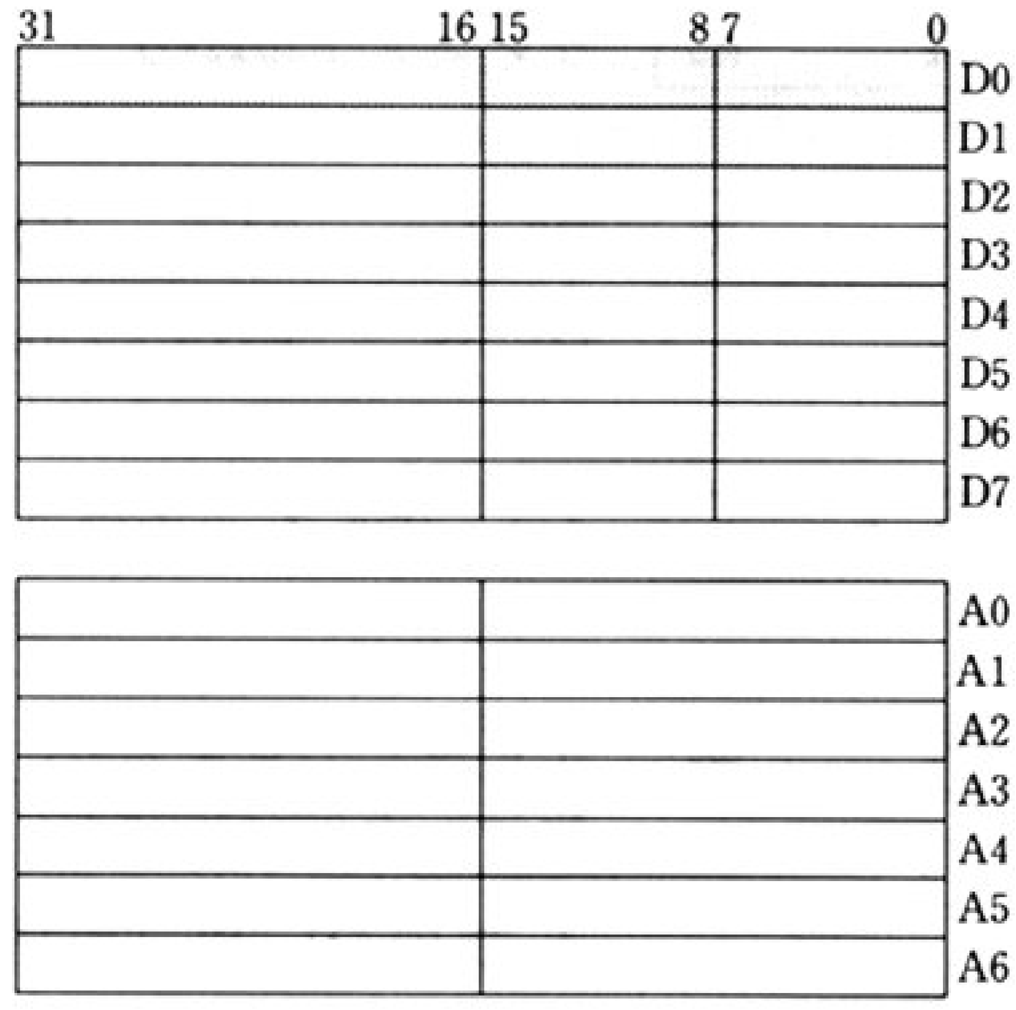
● B_EJECT　　IOCS コール番号：$4F（ハードディスク）

機　　能　シッピング（ハードディスク）

入　　力　D0. L　　$4F　IOCS コール番号
D1. W　　PDA×256（下位バイトは 0）

リターン　D0. L　　終了コード
D0. L ＜ 0……負数なら異常終了

解　　説　シッピングします。
入力の D1. W（PDA×256）は、ハードディスク関係のコールでは次のようになっています。



D1. W ＝ $8000……HARD DISK DRIVE0
⋮　　　⋮
D1. W ＝ $8F00……HARD DISK DRIVE15

プログラム例

```
*********************************************
* B_EJECT                                $4F
**** HDD ************************************

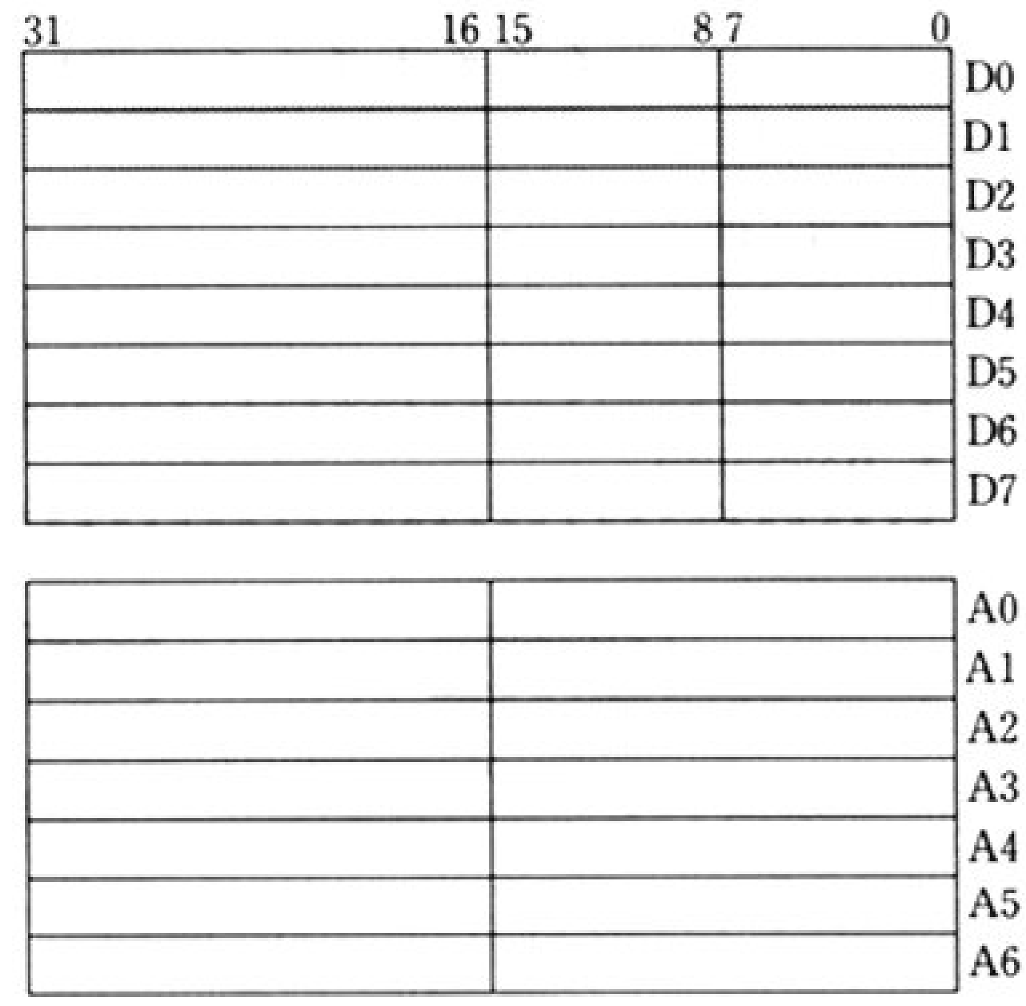
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_EJECT,d0     * IOCSコール番号をセット
        move.w  #$80_00,d1       * pda = $80
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

DATEBCD

● DATEBCD　　IOCSコール番号：$50



機　能　日付を時計にセットできる形式に変換

入　力　D0. L　$50　IOCSコール番号
D1. L　日付（バイナリ）

リターン　D0. L 変更後の日付データ(BCD)
D0. L ＝ −1…うるう年・大小の月判定におけるエラーが生じた

解　説　日付をバイナリ表現からBCD表現に変換します。
入力のD1. Lは変換前の日付です。
形式は次の通りです。

D1. L ＝ $0YYYMMDD
YY　＝ 1980～2079($7BC～$81F) …年（バイナリ）
MM ＝ 1～12($01～$0C) ……………月（バイナリ）
DD　＝ 1～31($01～$1F) ……………日（バイナリ）

リターンのD0. Lは、変換後の日付データ（BCD）です。
内容は次の通りです。

D0. L ＝ $UWYYMMDD
U　＝ 0～3……うるう年カウンタ
0……今年がうるう年
1……1年後がうるう年
2……2年後がうるう年
3……3年後がうるう年
W　＝ 0～6……曜日カウンタ
0……日曜日
⋮
6……土曜日
YY ＝ $00～$99……年BCD2桁
MM ＝ $01～$12……月BCD2桁

DD　= $01〜$31……日 BCD2 桁

うるう年カウンタは、年カウンタとともにインクリメントされます。

プログラム例

```
*******************************************
* DATEBCD                              $50
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

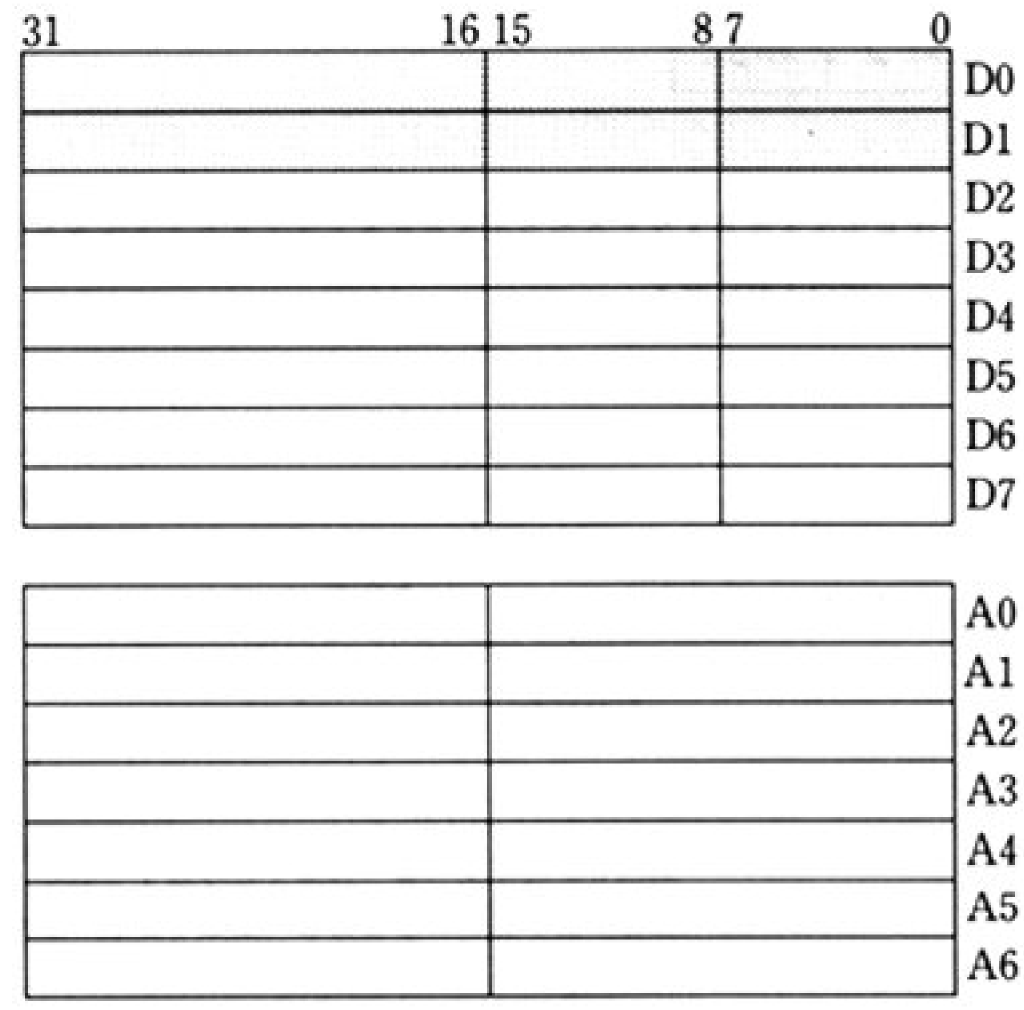
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DATEBCD,d0    * IOCSコール番号をセット
        move.l  #$07c6_03_0e,d1 * 1990年3月14日
        trap    #iocs           * IOCSコール

 * d0.l に BCD 表現の日付が返ってきます
 * d0.l = $23_90_03_14
        .
        .
        .end
```

DATESET

● DATESET　　IOCSコール番号：$51



機　能　日付設定

入　力　D0. L　$51　IOCSコール番号
D1. L　日付(BCD)

リターン　なし（D0. Lの内容は保証されません）

解　説　時計に日付を設定します。
ただし、正しい日付の指定であるかはチェックされません。
入力のD0. Lは設定する日付です。
内容は次の通りです。

YY　= 1980～2079($7BC～$81F) ………年（バイナリ）
MM = 1～12($01～$0C) …………………月（バイナリ）
DD　= 1～31($01～$1F) …………………日（バイナリ）

リターンのD0. Lは、変換後の日付データ（BCD）です。
内容は次の通りです。

D0. L = $UWYYMMDD
U　= 0～3……うるう年カウンタ
　0……今年がうるう年
　1……1年後がうるう年
　2……2年後がうるう年
　3……3年後がうるう年
W　= 0～6……曜日カウンタ
　0……日曜日
　⋮
　6……土曜日
YY　= $00～$99……年BCD2桁
MM = $01～$12……月BCD2桁
DD　= $01～$31……日BCD2桁
X68000のクロックIC (RTC)はBCD形式のデータを設定しなければなりませんので、DATEBCD ($50) でバイナリの日付データをBCDに変換した後、この

コールを使ってデータを設定します。

プログラム例

```
*******************************************
* DATESET                               $51
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

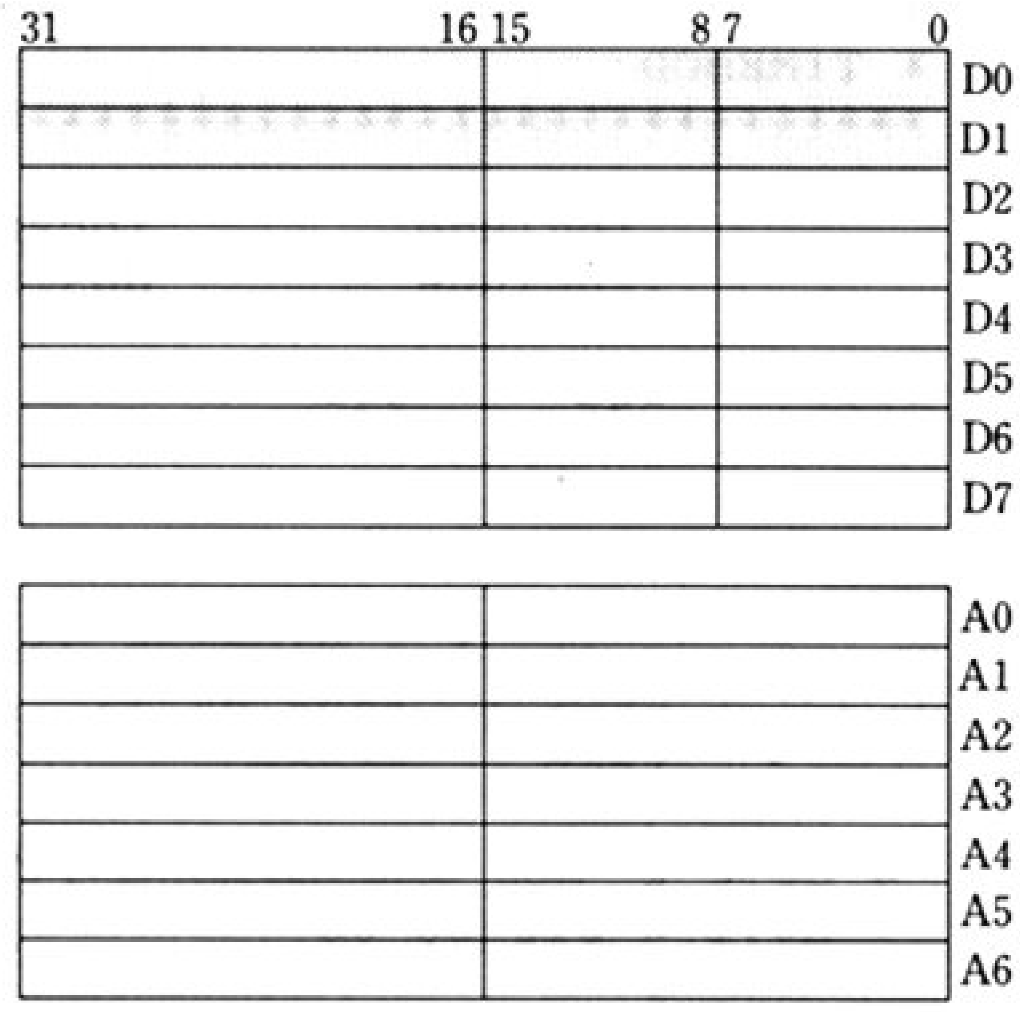
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DATESET,d0    * IOCSコール番号をセット
        move.l  #$23_90_03_14,d1* 1990年3月14日(水)
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

TIMEBCD

● TIMEBCD　　　IOCS コール番号：$52

機　能　時刻を時計にセットできる形式に変換



入　力　D0. L　$52　IOCS コール番号
D1. L　$00HHMMSS
時刻(バイナリ)

リターン　D0. L　変更後の時刻（BCD）
D0. L=$0THHMMSS
T　=1 システム既定（24 時間計であることを表す）
HH　=時 BCD2 桁……$00～$23
MM　=分 BCD2 桁……$00～$59
SS　=秒 BCD2 桁……$00～$59

解　説　時刻を時計にセットできる形式に変更します。
入力の D1. L は変換前の時刻です。
形式は次の通りです。

D1. L ＝ $00HHMMSS
HH ＝ 0～23（$00～$17）……時（バイナリ）
MM ＝ 0～59（$00～$3B）……分（バイナリ）
SS ＝ 0～59（$00～$3B）……秒（バイナリ）

プログラム例

```
*******************************************
* TIMEBCD                             $52
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

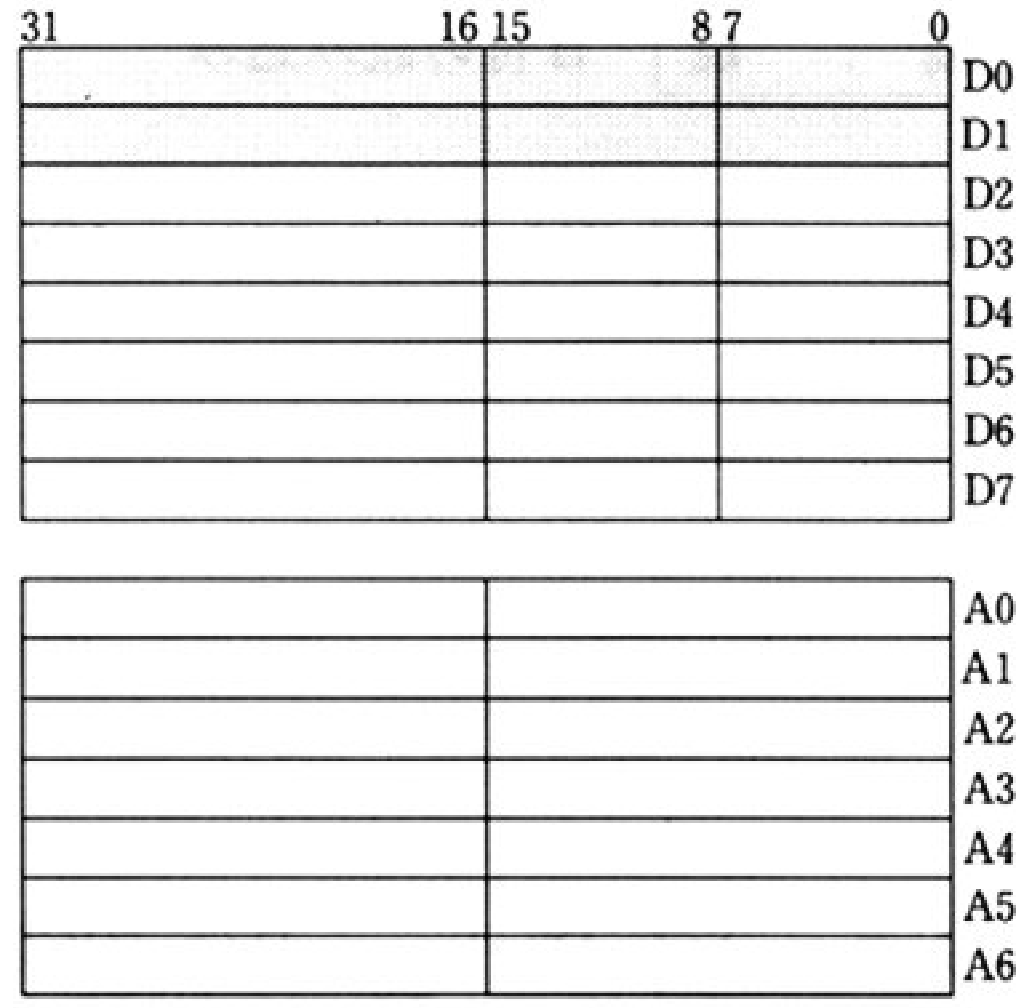
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TIMEBCD,d0    * IOCSコール番号をセット
        move.l  #$0c_05_1e      * hh = 12, mm = 05, ss = 30
        trap    #iocs           * IOCSコール

 * d0.l に BCD 表現の時刻が返ってきます
 * d0.l = $12_05_30
        .
        .
        .end
```

TIMESET

● TEMESET　　IOCS コール番号：$53



機　　能　時刻設定

入　　力　D0. L　$53　IOCS コール番号
D1. L　$0THHMMSS
時刻（BCD）

リターン　なし（D0. L の内容は保証されません）

解　　説　時計に時刻を設定します。
ただし、正しい時刻の指定であるかはチェックされません。
入力の D1. L は設定する時刻です。
形式は次の通りです。

D1. L ＝ $0THHMMSS
T　＝ 1 空白システム既定（24 時間計であることを表す）
HH　＝ $00～$23……時 BCD2 桁
MM　＝ $00～$59……分 BCD2 桁
SS　＝ $00～$59……秒 BCD2 桁

プログラム例

```
*********************************************
* TIMESET                               $53
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

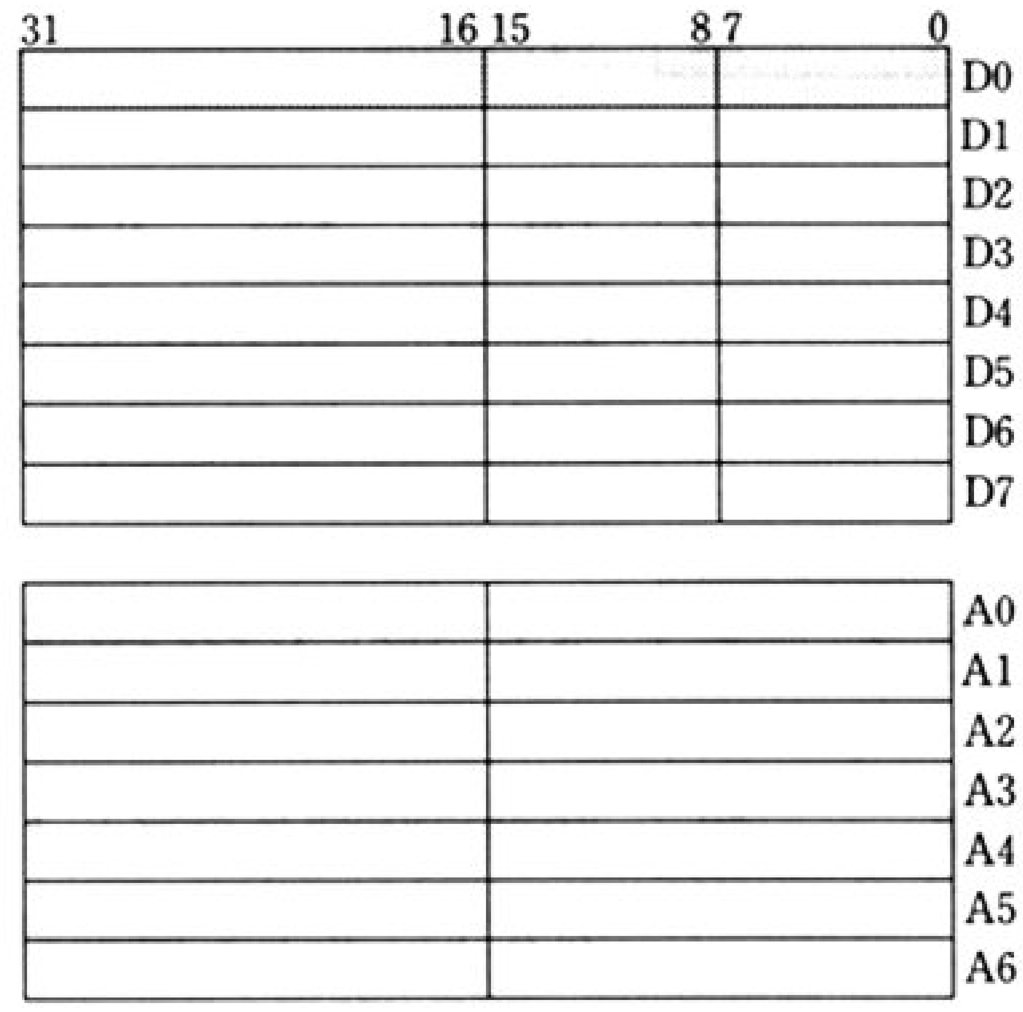
        .text
        .
        .
        moveq.l #_TIMESET,d0     * IOCSコール番号をセット
        move.l  #$01_12_05_30,d1* 12:05:30
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

DATEGET

● DATEGET　　　IOCS コール番号：$54

機　　能　日付の読み込み

入　　力　D0. L　　$54　　IOCS コール番号



リターン　D0. L　　日付（BCD）

D0. L ＝ $0WYYMMDD

W　　＝ 0～6…………曜日カウンタ（日曜日 ＝ 0、土曜日 ＝ 6）

YY　＝ $00～$99……年 BCD2 桁（1980 年～2079 年）

MM　＝ $01～$12……月 BCD2 桁

DD　＝ $01～$31……日 BCD2 桁

解　　説　時計から日付を読み込みます。

プログラム例

```
********************************************
* DATEGET                               $54
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

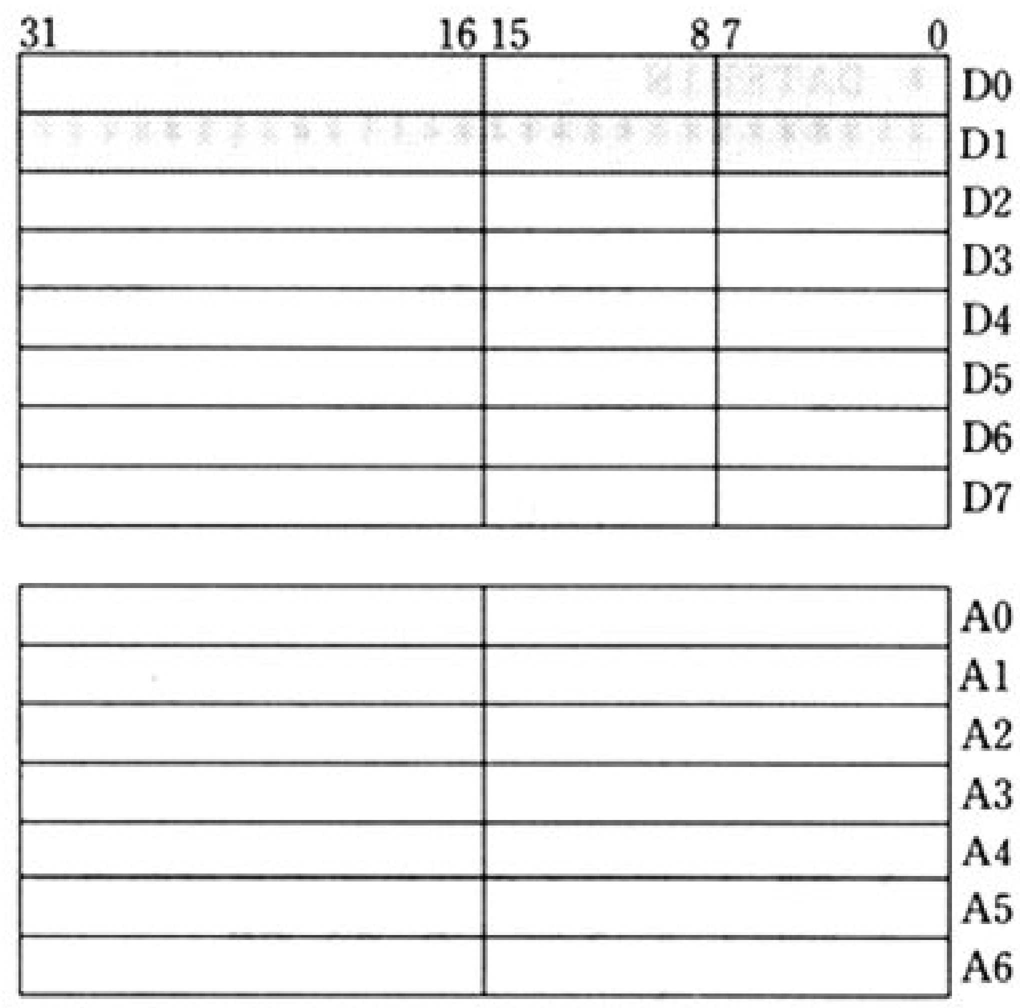
        .text
        .
        .
        moveq.l #_DATEGET,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール

        * d0.l に BCD 表現で現在の日付が返ってきます
        .
        .
        .end
```

DATEBIN

● DATEBIN　　IOCS コール番号：$55

機　能　日付を BCD 表現からバイナリ表現に変換



入　力　D0. L　$55　IOCS コール番号
D1. L　日付（BCD 表現）

リターン　D0. L　変換後の日付（バイナリ）

D0. L =$WYYYMMDD

W	=	0〜6……………………………曜日
YYY	=	1980〜2079（$7BC〜$81F)…年(バイナリ)
MM	=	01〜12（$01〜$0C)…………月(バイナリ)
DD	=	01〜31（$01〜$1F)…………日(バイナリ)

解　説　日付を BCD 表現からバイナリ表現に変換します。
入力の D1. L は変換前のデータで、形式は次の通りです。

D1. L = $0WYYMMDD

W	=	0〜6…………曜日カウンタ（日曜日=0、土曜日=6)
YY	=	$00〜$99……年 BCD2 桁（1980 年〜2079 年)
MM	=	$01〜$12……月 BCD2 桁 01〜12
DD	=	$01〜$31……日 BCD2 桁 01〜31

プログラム例

```
****************************************
* DATEBIN                            $55
****************************************

        .include        iocscall.mac
        .include        doscall.mac

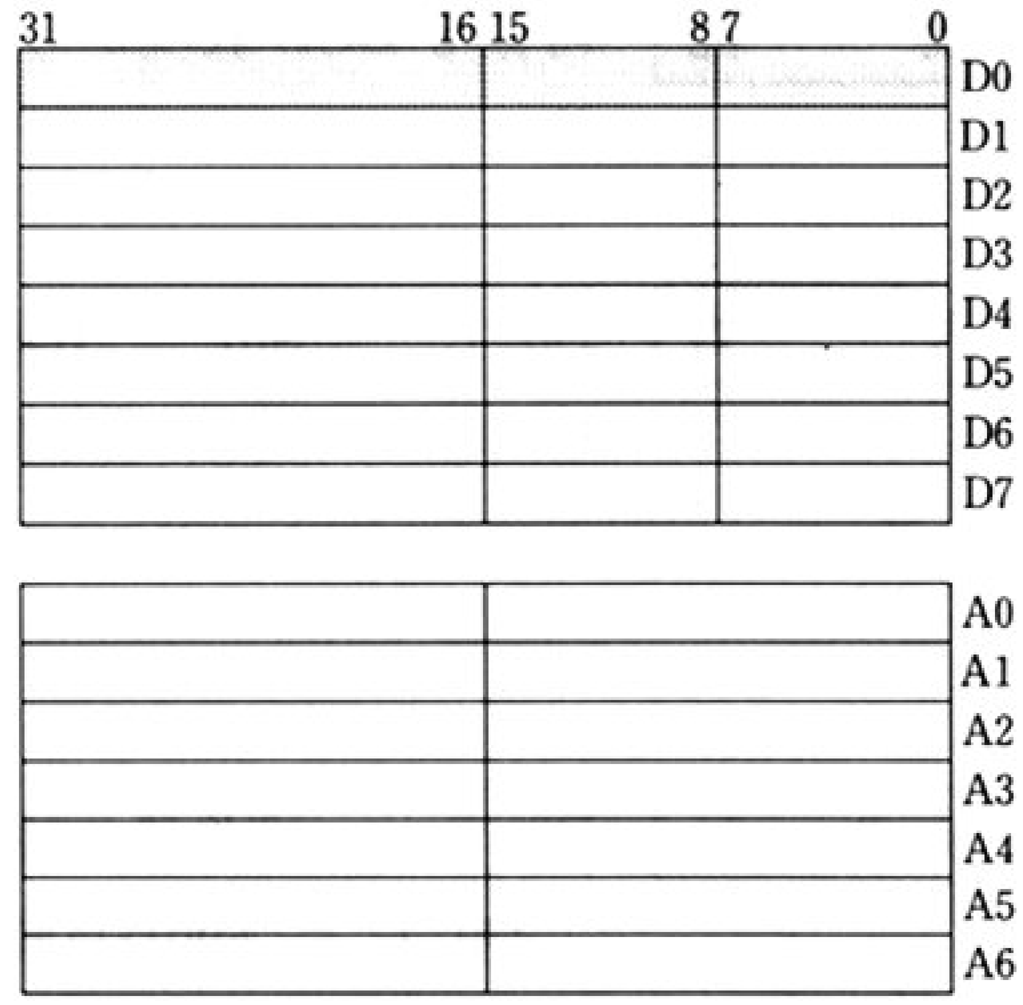
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DATEBIN,d0    * IOCSコール番号をセット
        move.l  #$03_90_03_14,d1* 1990年03月14日(水)
        trap    #iocs           * IOCSコール

 * d0.l にバイナリ形式の日付が返ってきます
 * d0.l = $37_c6_03_0e
        .
        .
        .end
```

TIMEGET

● TIMEGET　　IOCS コール番号：$56



機　　能　時刻の読み込み

入　　力　D0. L　　$56　　IOCS コール番号

リターン　D0. L　読み込んだ時刻のデータ（BCD）

D0. L ＝ $00HHMMSS

HH　＝ $00～$23……時 BCD2 桁

MM　＝ $00～$59……分 BCD2 桁

SS　＝ $00～$59……秒 BCD2 桁

解　　説　時計から時刻を読み込みます。

プログラム例

```
********************************************
* TIMEGET                               $56
********************************************

        .include        iocscall.mac
        .include        doscall.mac

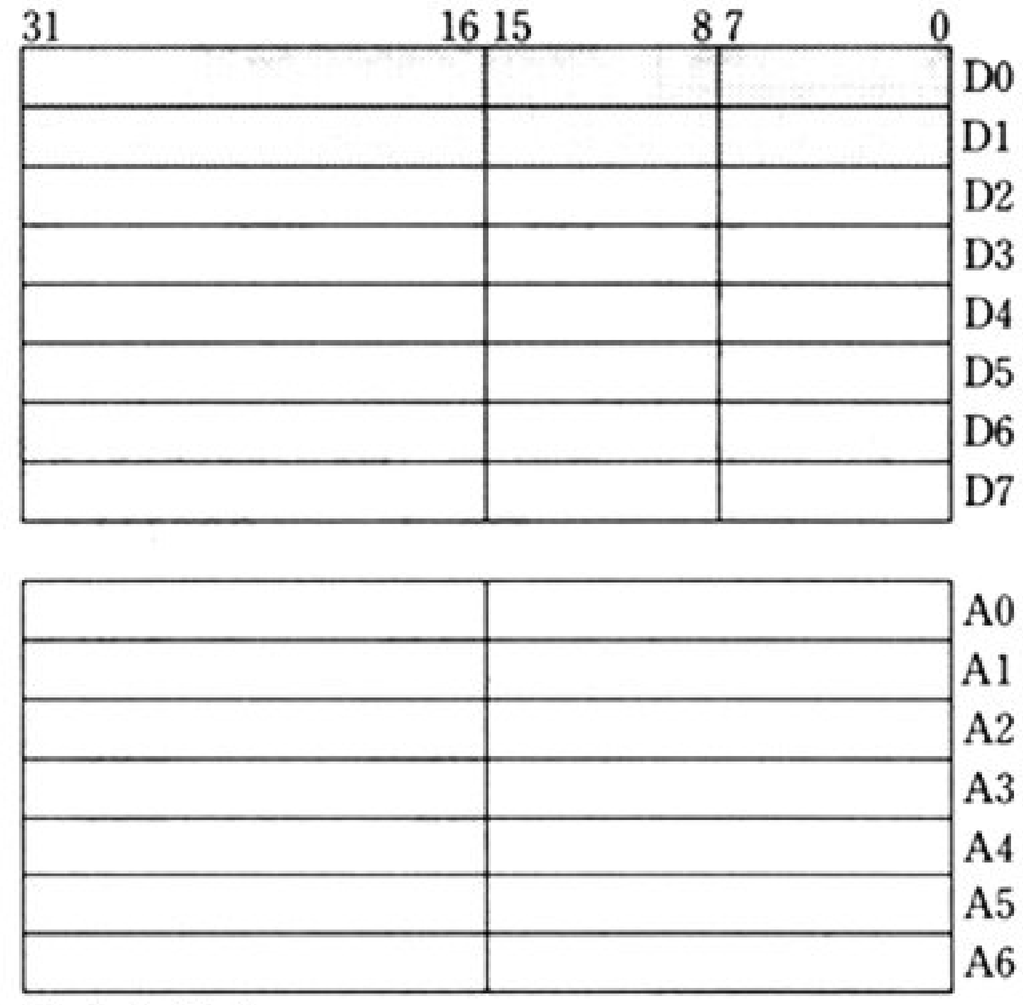
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TIMEGET,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール

 * d0.l に BCD 表現で現在の時刻が返ってきます
        .
        .
        .end
```

TIMEBIN

● TIMEBIN　　IOCS コール番号：$57



機　　能　時刻を BCD 表現からバイナリ表現に変換

入　　力　D0. L　　$57　　IOCS コール番号
D1. L　　時刻（BCD 表現）

リターン　D0. L　　変換後の時刻（バイナリ）

D0. L ＝ $00HHMMSS
HH　＝ 0～23($00～$17) ……時(バイナリ)
MM　＝ 0～59($00～$3B) ……分(バイナリ)
SS　＝ 0～59($00～$3B) ……秒(バイナリ)

解　　説　時刻を BCD 表現からバイナリ表現に変換します。
入力の D1. L は変換前のデータで、形式は次の通りです。

D1. L ＝ $00HHMMSS
HH　＝ $00～$23……時 BCD2 桁
MM　＝ $00～$59……分 BCD2 桁
SS　＝ $00～$59……秒 BCD2 桁

プログラム例

```
********************************************
* TIMEBIN                               $57
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

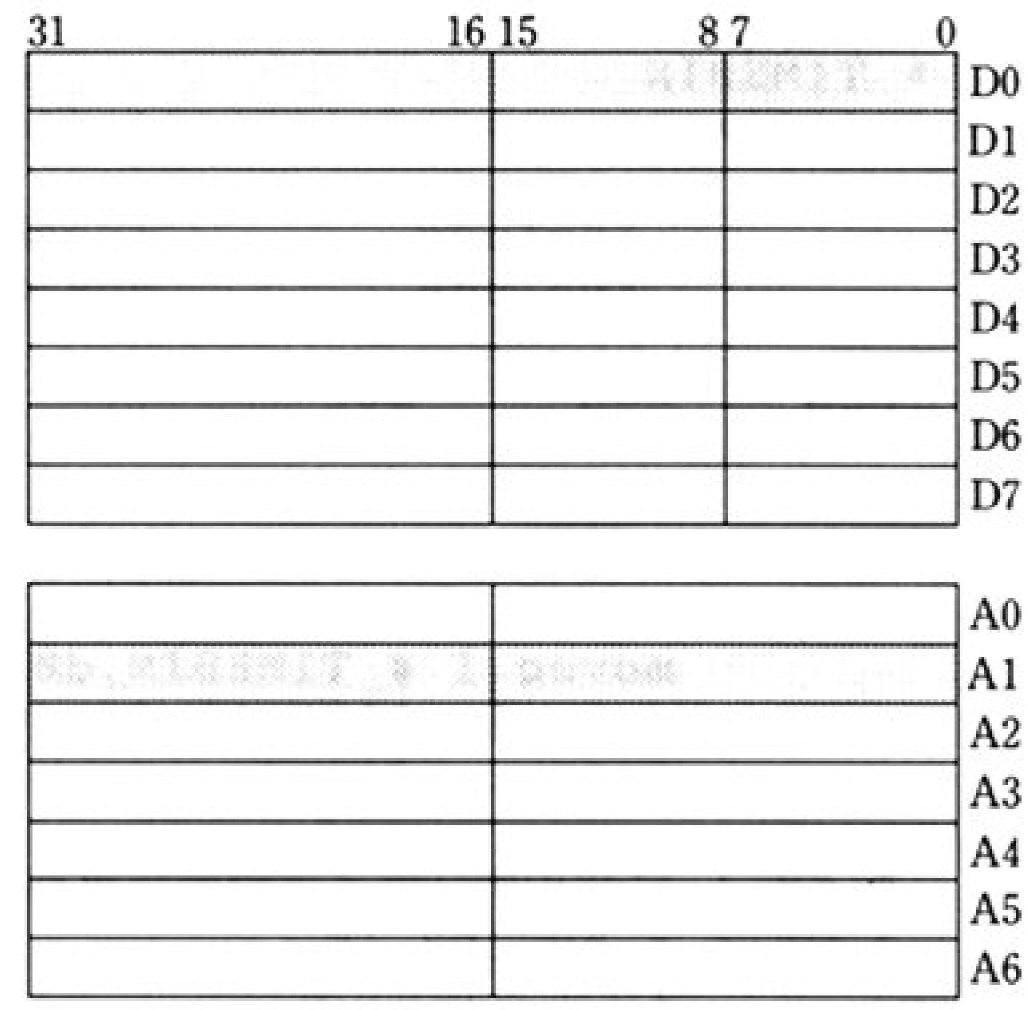
        .text
        .
        .
        moveq.l #_TIMEBIN,d0     * IOCSコール番号をセット
        move.l  #$00_12_05_30,d1* 12:05:30
        trap    #iocs            * IOCSコール

 * d0.l にバイナリ形式で時刻が返ってきます
 * d0.l = $0c_05_1e
        .
        .
        .end
```

DATECNV

● DATECNV　　IOCS コール番号：$58

機　　能　日付を表す文字列をバイナリデータに変換



入　　力　D0. L　$58　IOCS コール番号
A1. L　日付を表す文字列の先頭アドレス

リターン　D0. L　日付（バイナリ）

D0. L ＝ $0YYYMMDD
YYY　＝ 年(バイナリ)……1980～2079（$7BC～$81F)
MM　＝ 月(バイナリ)……1～12（$01～$0C)
DD　＝ 日(バイナリ)……1～31（$01～$1F)
D0. L ＝ －1……異常終了（文字列の書式が異常）

解　　説　日付を表す文字列を、バイナリデータに変換します。
区切り記号は、'／'でも'－'でもかまいません。
うるう年・大小の月判定はしません。
スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
入力の A1. L は、日付を表す文字列の先頭アドレスです。
文字列は次の形式で表します。

(A1) ＝ '1990／03／14',0
または、
(A1) ＝ '90／03／14',0

プログラム例

```
*****************************************
* DATECNV                           $58
*****************************************

        .include        iocscall.mac
        .include        doscall.mac

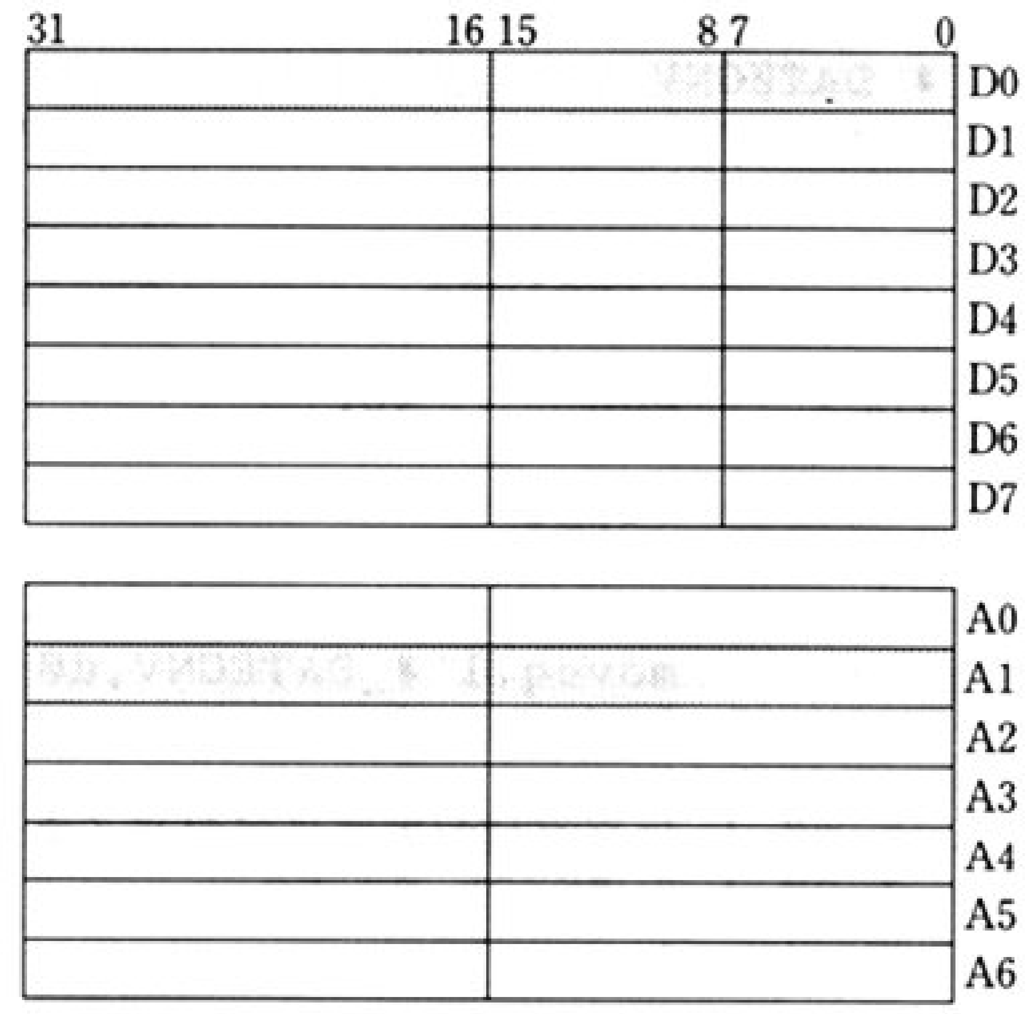
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DATECNV,d0     * IOCSコール番号をセット
        lea     date,a1          * 日付を表す文字列のアドレス
        trap    #iocs            * IOCSコール
 * d0.l に文字列が表す日付がバイナリ形式で返ってきます
 * d0.l = $07c6_03_0e
        .
        .
        .data
date:   dc.b    "1990/03/14",0
        .
        .
        .end
```

TIMECNV

● TIMECNV　　　IOCS コール番号：$59

機　能	時刻を表す文字列をバイナリデータに変換
入　力	D0. L　　$59　　IOCS コール番号 A1. L　　時刻を表す文字列の先頭アドレス
リターン	D0. L　　時刻（バイナリ）



D0. L ＝ $00HHMMSS

HH　　＝ 00～23($00～$17)……時(バイナリ)

MM　　＝ 00～59($00～$3B)……分(バイナリ)

SS　　＝ 00～59($00～$3B)……秒(バイナリ)

D0. L ＝ −1……異常終了（文字列の書式が異常）

解　説

時刻を表す文字列を、バイナリデータに変換します。

区切り記号は、'：'を使用します。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力の A1. L は、時刻を表す文字列の先頭アドレスです。

文字列は次の形式で表します。

(A1) ＝ '12：05：30',0

プログラム例

```
*******************************************
* TIMECNV                              $59
*******************************************

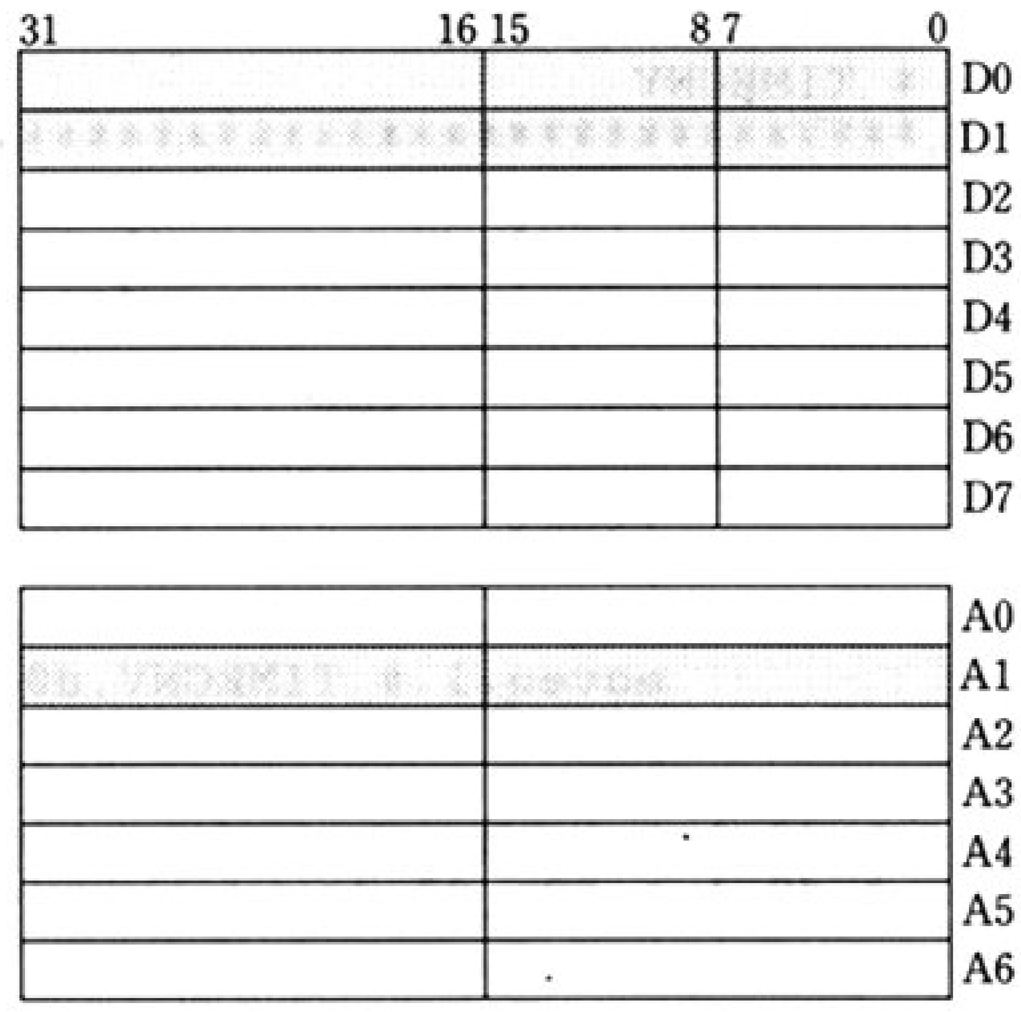
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TIMECNV,d0    * IOCSコール番号をセット
        lea     time,a1         * 日付を表す文字列のアドレス
        trap    #iocs           * IOCSコール
 * d0.l に文字列が表す時刻がバイナリ形式で返ってきます
 * d0.l = $0c_05_1e
        .
        .
        .data
time:   dc.b    "12:05:30",0
        .
        .
        .end
```

DATEASC

● DATEASC　　IOCS コール番号：$5A



機　能　バイナリデータを日付を表す文字列に変換します

入　力

D0. L	$5A	IOCS コール番号
D1. L		日付のバイナリデータ
A1. L		変換後の日付を表す文字列の先頭アドレス

リターン

D0. L	エラーなら－1、正常終了なら不定
A1. L	変換後の日付を表す文字列の終端（ヌル文字）のアドレス

解　説　バイナリデータを、日付を表す文字列に変換します。
スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
入力 D1. L は、変換前のデータで内容は次の通りです。

```
D1. L  = $FYYYMMDD
  F      = 0～3 文字列フォーマット
           0……1990／03／14
           1……1990—03—14
           2……90／03／14
           3……90—03—14
  YYY    = 1980～2079 ($7BC～$81F) …… (バイナリ)
  MM     = 01～12 ($01～$0C) ………… (バイナリ)
  DD     = 01～31 ($01～$1F) ………… (バイナリ)
```

入力の A1. L は、変換後の日付を表す文字列の先頭アドレスです。
変換後の文字列は次の形式です。

(A1) = '1990／03／14', 0
または、
(A1) = '90／03／14', 0
この文字列領域は、11 バイト以上確保する必要があります。

プログラム例

```
********************************************
* DATEASC                               $5A
********************************************

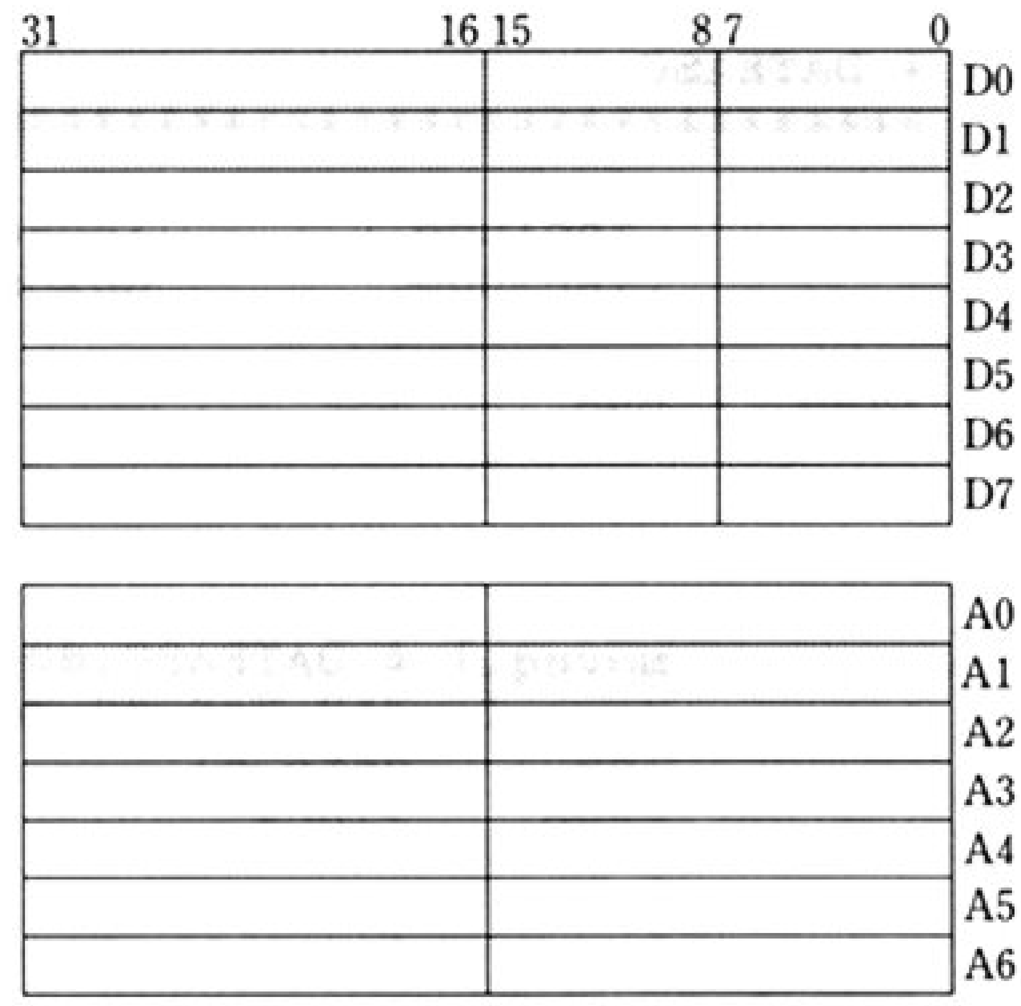
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DATEASC,d0    * IOCSコール番号をセット
        move.l  #$2_7c6_03_0e,d1* yy = 1990, mm = 03, dd = 14
        lea     date,a1         * 日付を表す文字列を格納するアドレス
        trap    #iocs           * IOCSコール
        .
        .                       * (a1) に
                                * 時刻を表す文字列が返る
                                * (a1) = "90/03/14",0
        .
        .data
date:   ds.b    12              * 12 バイトのエリアを確保
        .
        .
        .end
```

TIMEASC

● TIMEASC　　　IOCS コール番号：$5B



機　　能　バイナリデータを時刻を表す文字列に変換

入　　力

D0. L	$5B	IOCS コール番号
D1. L	時刻のバイナリデータ	
A1. L	変換後の時刻を表す文字列の先頭アドレス	

リターン

D0. L	エラーなら−1、そうでなければ不定
A1. L	変換の時刻を表す文字列の終端（ヌル文字）のアドレス

解　　説　バイナリデータを、時刻を表す文字列に変換します。
スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。
入力の D1. L は、変換前のデータで内容は次の通りです。

D1. L ＝ $00HHMMSS
　HH　＝ 0〜23（$00〜$17）……時（バイナリ）
　MM　＝ 0〜59（$00〜$3B）……分（バイナリ）
　SS　＝ 0〜59（$00〜$3B）……秒（バイナリ）

入力の A1. L は、変換後の時刻を表す文字列の先頭アドレスです。
変換後の文字列は次の形式です。

(A1) ＝ '12：05：30', 0
この文字列領域は 9 バイト以上確保する必要があります。

プログラム例

```
*******************************************
* TIMEASC                              $5B
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

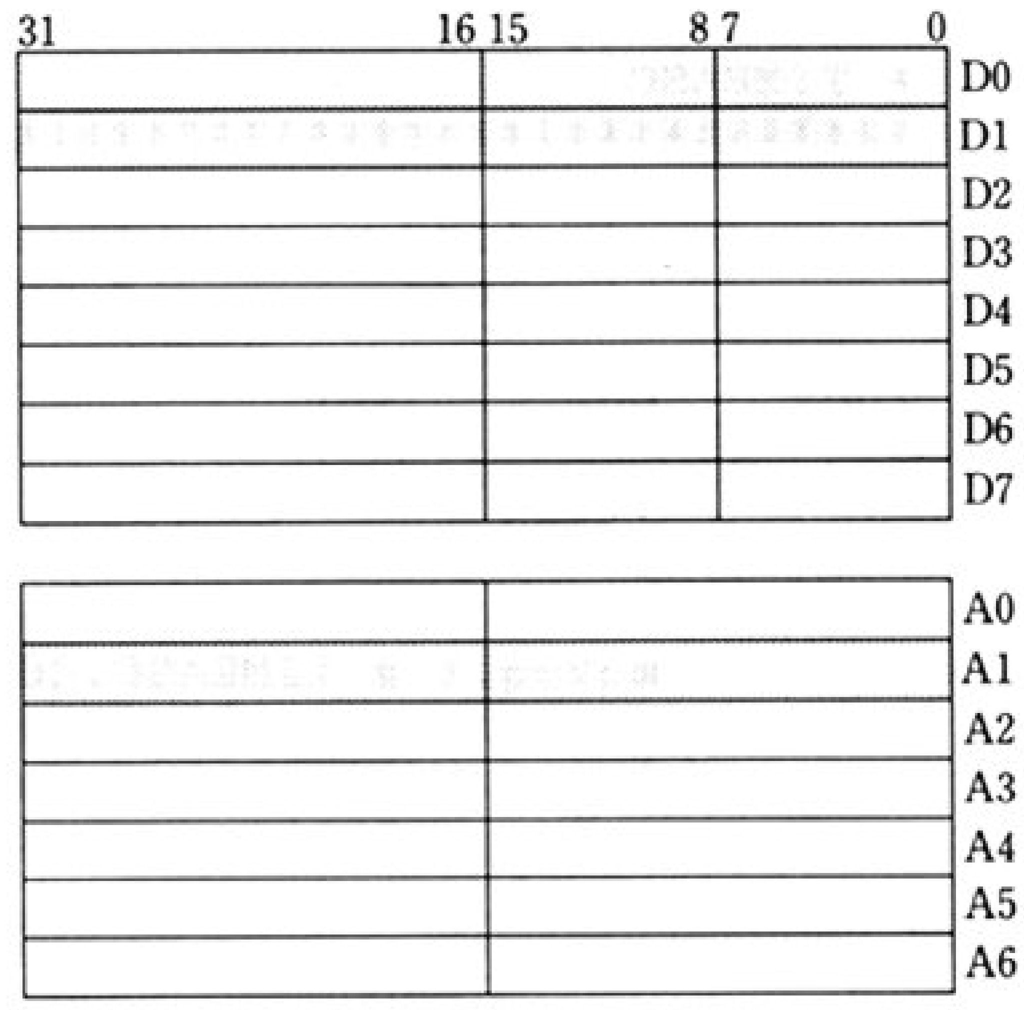
        .text
        .
        .
        moveq.l #_TIMEASC,d0    * IOCSコール番号をセット
        move.l  #$0c_05_1e,d1   * hh = 12, mm = 5, ss = 30
        lea     time,a1         * 文字列を格納するアドレス
        trap    #iocs           * IOCSコール

                                * (a1) に
                                * 時刻を表す文字列が返る
                                * (a1) = "12:05:30",0
        .
        .data
time:   ds.b    10              * 10 バイトのエリアを確保
        .
        .
        .end
```

DAYASC

● DAYASC　　IOCS コール番号：$5C

機　　能　バイナリデータを曜日を表す文字列に変換



入　　力

D0. L	$5C	IOCS コール番号
D1. L	曜日（バイナリ）	
A1. L	変換後の曜日を表す文字列の先頭アドレス	

リターン

(A1)	変換後の曜日を表す文字列 '月',0

解　　説　バイナリデータを、曜日を表す文字列に変換します。

スーパーバイザ領域のアドレスも指定できますので、不正なアドレスを指定しないように注意してください。

入力のD1. Lは、曜日を表すバイナリデータです(0〜6日曜日＝0、…、土曜日＝6)。

入力の A1. L は、変換後の曜日を表す文字列の先頭アドレスです。

この文字列領域は、3 バイト以上確保する必要があります。

プログラム例

```
*******************************************
* DAYASC                              $5C
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

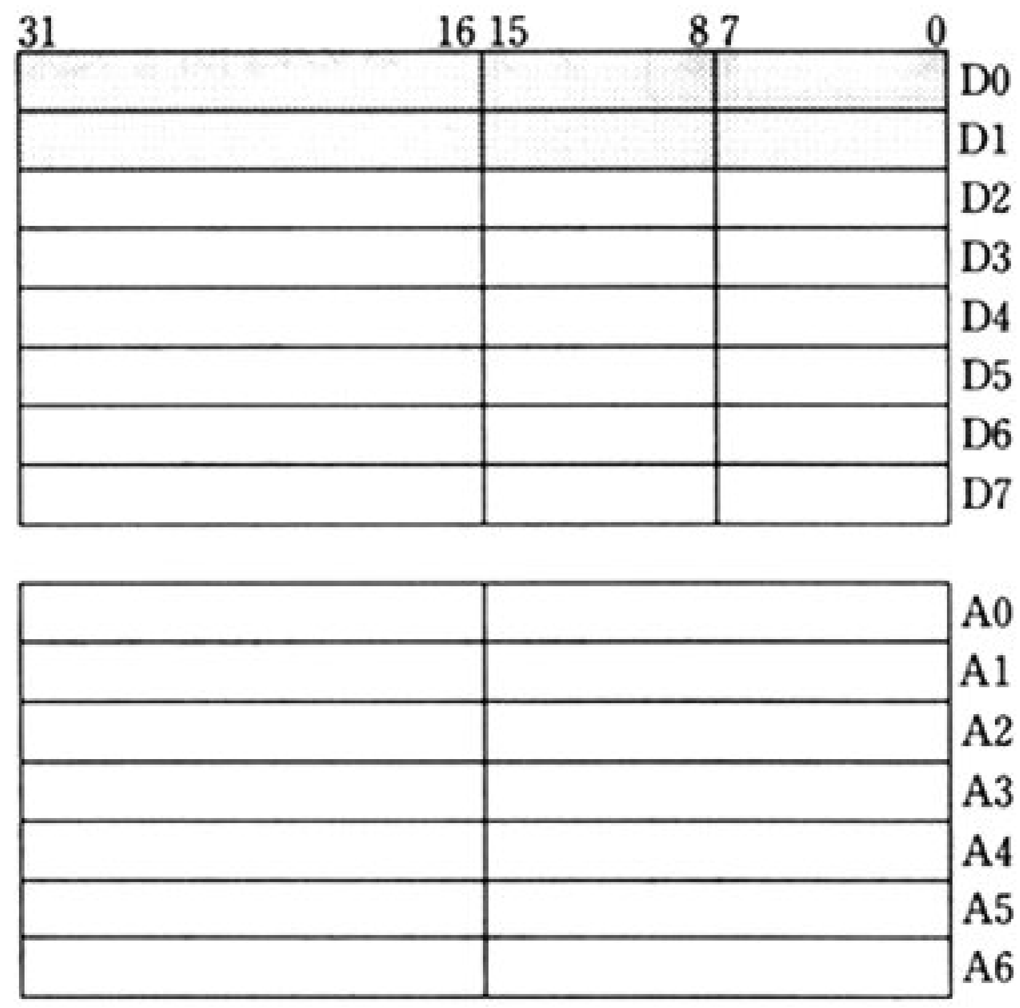
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DAYASC,d0     * IOCSコール番号をセット
        moveq.l #1,d1           * 月曜日はバイナリ形式で 1
        lea     day,a1          * 曜日を表す文字列を格納するアドレス
        trap    #iocs           * IOCSコール

 * (a1) に 曜日を表す文字列が返ってきます
        .
        .
        .data
day:    ds.b    3               * 3 バイトのエリアを確保
        .
        .
        .end
```

ALARMMOD

● ALARMMOD　　IOCS コール番号：$5D



機　能　アラームの禁止／許可

入　力　D0. L　　$5D　IOCS コール番号
D1. L　　設定値

リターン　D0. L　　状態
D0. L　＝ 0……禁止
D0. L　＝ 1……許可

解　説　アラームの禁止／許可を設定します。
入力の D1. L は設定値です。
内容は次の通りです。

D1. L ＝ 0…………禁止
D1. L ＝ 1…………許可
D1. L ＝ 2…………状態を見る

プログラム例

```
*********************************************
* ALARMMOD                                $5D
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

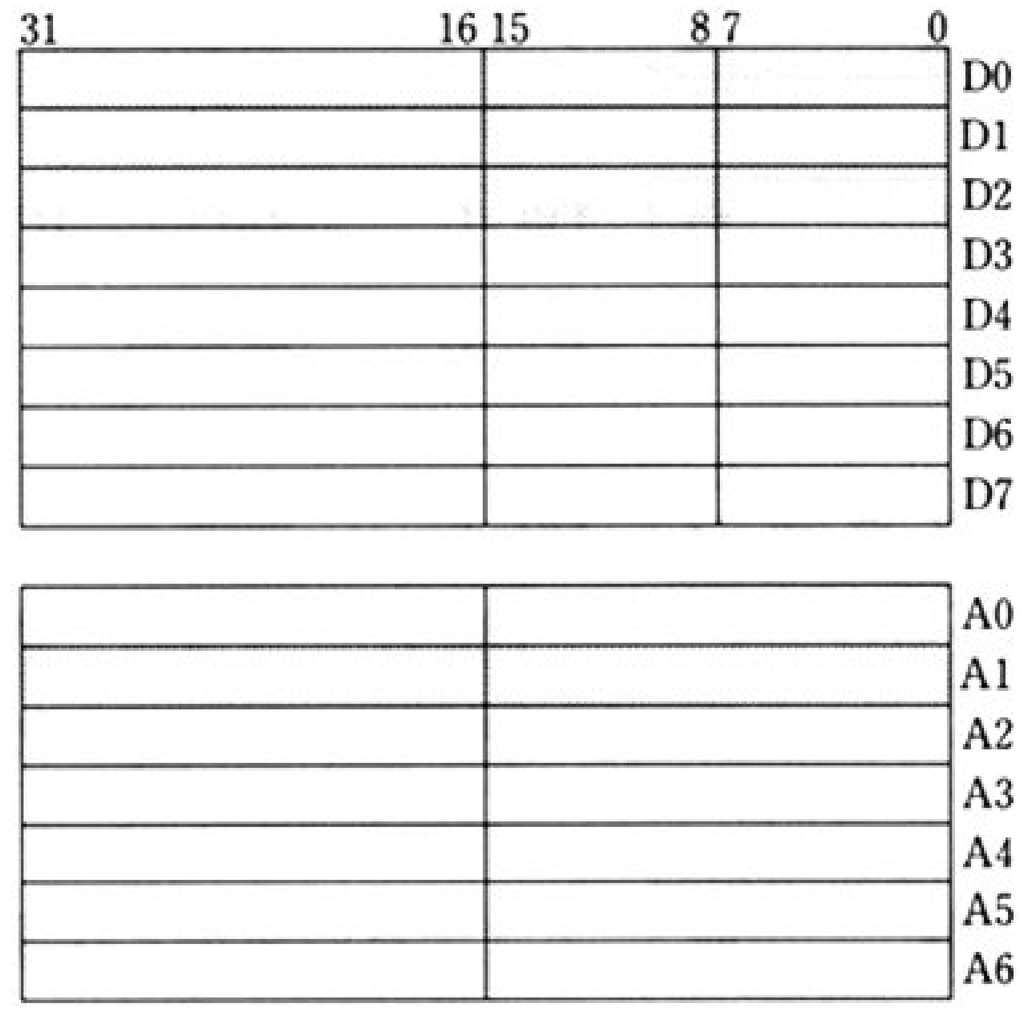
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ALARMMOD,d0   * IOCSコール番号をセット
        moveq.l #1,d1           * アラーム許可
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

ALARMSET

● ALARMSET　　IOCS コール番号：$5E

機　　能	アラームの時間と処理アドレスの設定



入　　力

D0. L	$5E	IOCS コール番号
A1. L	テレビコードまたは処理アドレス	
D1. L	アラーム時間	
D2. L	OFF されるまでの時間	

リターン　D0. L　　終了コード

D0. L ＝ −1……エラー

解　　説　アラームの時間と処理アドレスを設定します。

入力の A1. L はテレビコントロールまたは処理アドレスで内容は次の通りです。

A1. L ＝ 0 …………パワーオンして、テレビオン&コンピュータモードにする

A1. L ＝ −1 ………パワーオンのみ、テレビコントロールしない

A1, L ＝ 1～$3F……テレビコントロールする（TVCTRL（$0C；テレビコントロールを行う）を参照）

A1. L ＝ 上記以外…処理アドレス
この場合、偶数でかつ、処理アドレスのプログラムの先頭は$60（BRA 命令）である必要があります。

入力の D1. L はアラームの時間です。
内容は次の通りです。

D1. L　＝ $0WDDHHMM

W　＝ $00～$06……曜日（日曜日 ＝ 0、…、土曜日 ＝ 6）
DD　＝ $01～$31……日付 BCD2 桁
HH　＝ $00～$23………時 BCD2 桁
MM　＝ $00～$59………分 BCD2 桁

曜日に$F を、日時分に$FF を指定すると、それぞれが無指定になります。
ただし、すべてを無指定にしないでください。
入力の D2. L は、テレビあるいはコンピュータがアラームでパワーオンされてから、パワーオフされるまでの時間（1 分単位）です。
ただし 0 を指定した場合、いつまでもパワーオフしません。

アラームでパワーオンされるのは、本体前面のスイッチがオフになっていて、POWER LEDが赤く点灯しているときのみです。

プログラム例

```
*******************************************
* ALARMSET                             $5E
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

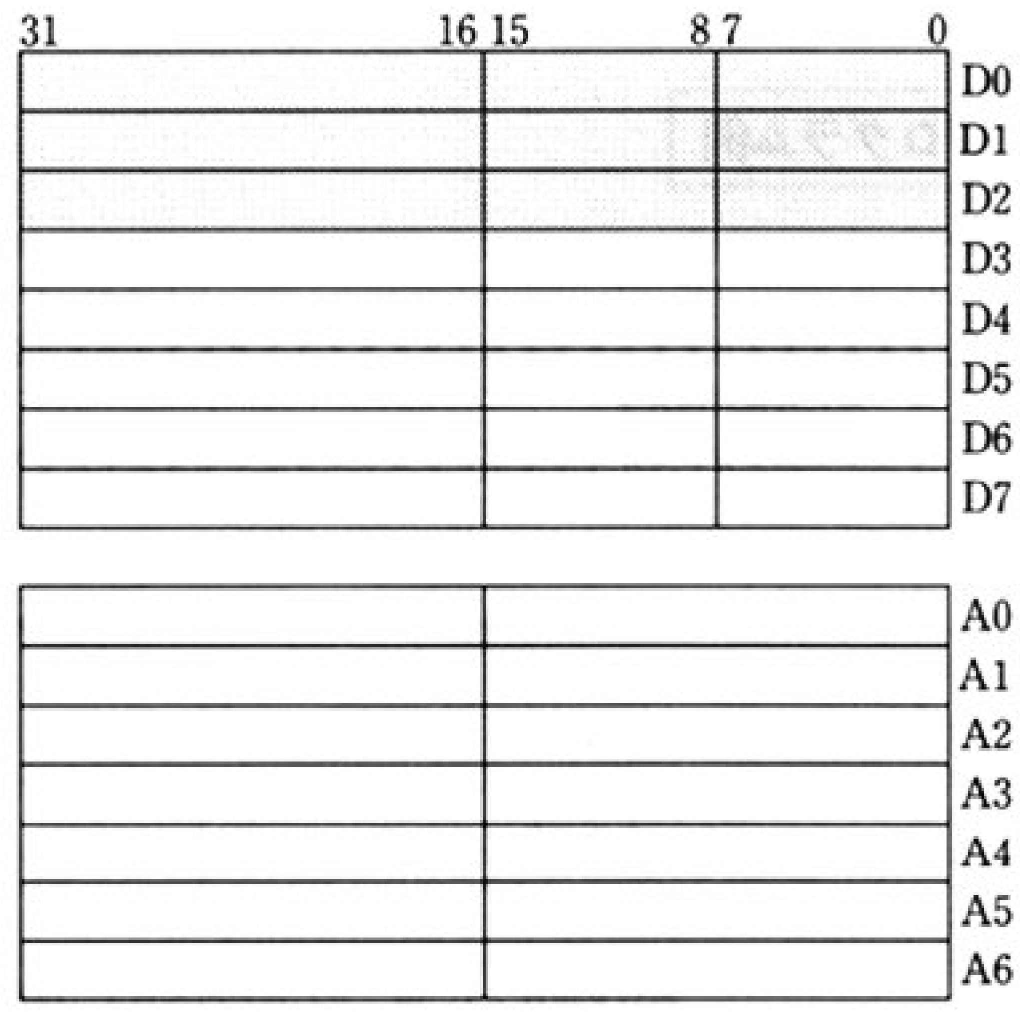
        .text
        .
        .
        moveq.l #_ALARMSET,d0    * IOCSコール番号をセット
        lea     func,a1          * 処理アドレス
        move.l  #$02_18_12_00,d1* Tue. 18th 12:00 にセット
        move.l  #30,d2           * 30分間電源 ON
        trap    #iocs            * IOCSコール
        .
        .
func:
        bra     func0

func0:
        .
        .
        rts
        .
        .
        .end
```

ALARMGET

● ALARMGET　　IOCS コール番号：$5F

機　能　アラームの時間と処理アドレスの読み込み



入　力　D0. L　　$5F　　IOCS コール番号

リターン　D0. L　　処理アドレス

D1. L　　アラームの時間

D1. L ＝ $0WDDHHMM

W　＝ 0～6……曜日

DD　＝ $01～$31……日付 BCD2 桁

HH　＝ $00～$23……時 BCD2 桁

MM　＝ $00～$59……分 BCD2 桁

曜日が$F で、日時分が$FF になっている場合は無指定を表します。

D2. L　　テレビまたはコンピュータがオフされるまでの時間（1 分単位）

解　説　アラームの時間と処理アドレス（ALARMSET（$SE）参照）を読み込みます。

プログラム例

```
*********************************************
* ALARMGET                                $5F
*********************************************

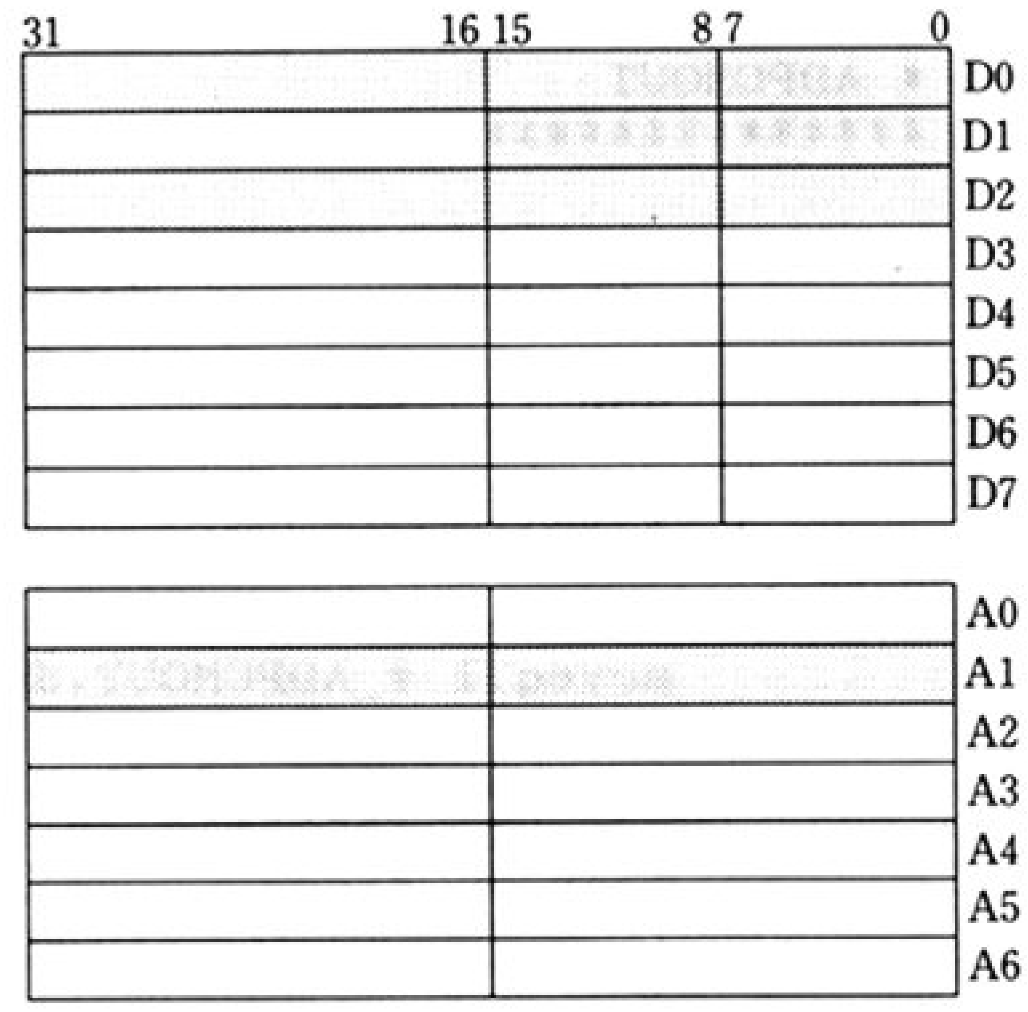
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ALARMGET,d0    * IOCSコール番号をセット
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

ADPCMOUT

● ADPCMOUT　　IOCS コール番号：$60



機　能　ADPCM データ出力

入　力

D0. L	$60	IOCS コール番号
A1. L	出力データアドレス	
D1. W	サンプリング周波数×256＋出力モード	
D2. L	出力データの長さ	

リターン　なし（D0. L の内容は保証されません）

解　説　ADPCM へデータを出力します。

長さが$FEFF 以下の場合は、出力を開始してすぐリターンします。

長さが$FF00 以上の場合は、内部で$FF00 単位で処理しますので、すぐリターンできません。

入力の A1. L は出力データアドレスです。

入力の D1. W は次のようになっています。

ビット15～8：サンプリング周波数
0……3.9KHz（1秒につき 1950 バイト）
1……5.2KHz（1秒につき 2600 バイト）
2……7.8KHz（1秒につき 3900 バイト）
3……10.4KHz（1秒につき 5200 バイト）
4……15.6KHz（1秒につき 7800 バイト）

ビット7～0 ：出力モード
0……音声出力カット
1……音声出力左
2……音声出力右
3……音声出力左右

入力の D2. L は出力データの長さです。

プログラム例

```
*******************************************
* ADPCMOUT                             $60
*******************************************

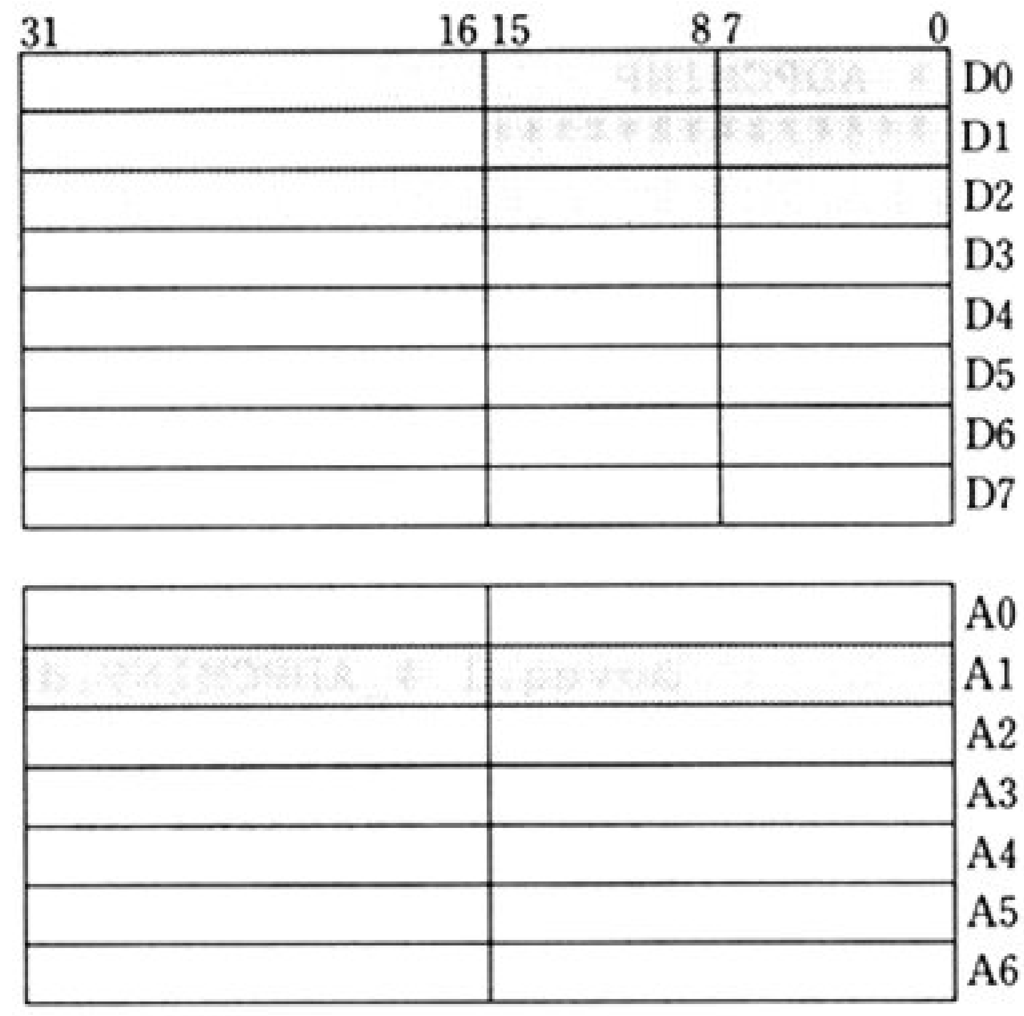
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        mcveq.l #_ADPCMOUT,d0   * IOCSコール番号をセット
        lea     pcmdat,a1       * データアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        move.l  #7800*10,d2     * 10 秒間のデータが pcmdat にあります
        trap    #iocs           * IOCSコール
        .
        .
        .data
pcmdat:
        ds.b 7800*10            * 出力データ
        .
        .
        .end
```

ADPCMINP

● ADPCMINP　　IOCSコール番号：$61



機　　能　ADPCMデータ入力

入　　力

D0. L	$61	IOCSコール番号
A1. L	入力データアドレス	
D1. W	サンプリング周波数×256＋出力モード	
D2. L	入力データの長さ	

リターン　なし（D0. Lの内容は保証されません）

解　　説　ADPCMからデータを入力します。

長さが$FF00以上の場合は、内部で$FF00単位で処理しますので、すぐリターンできません。

入力のA1. Lは、入力データアドレスです。

入力のD1. Wは次のようになっています。

ビット15〜8：サンプリング周波数
- 0……3.9KHz（1秒につき1950バイト）
- 1……5.2KHz（1秒につき2600バイト）
- 2……7.8KHz（1秒につき3900バイト）
- 3……10.4KHz（1秒につき5200バイト）
- 4……15.6KHz（1秒につき7800バイト）

ビット7〜0　：出力モード
- 0……音声出力カット
- 1……音声出力左
- 2……音声出力右
- 3……音声出力左右

データ入力はモノラルで入力しますが、モニター出力を出力モードで指定できます。

入力のD2. Lは、入力データの長さです。

プログラム例

```
*******************************************
* ADPCMINP                             $61
*******************************************

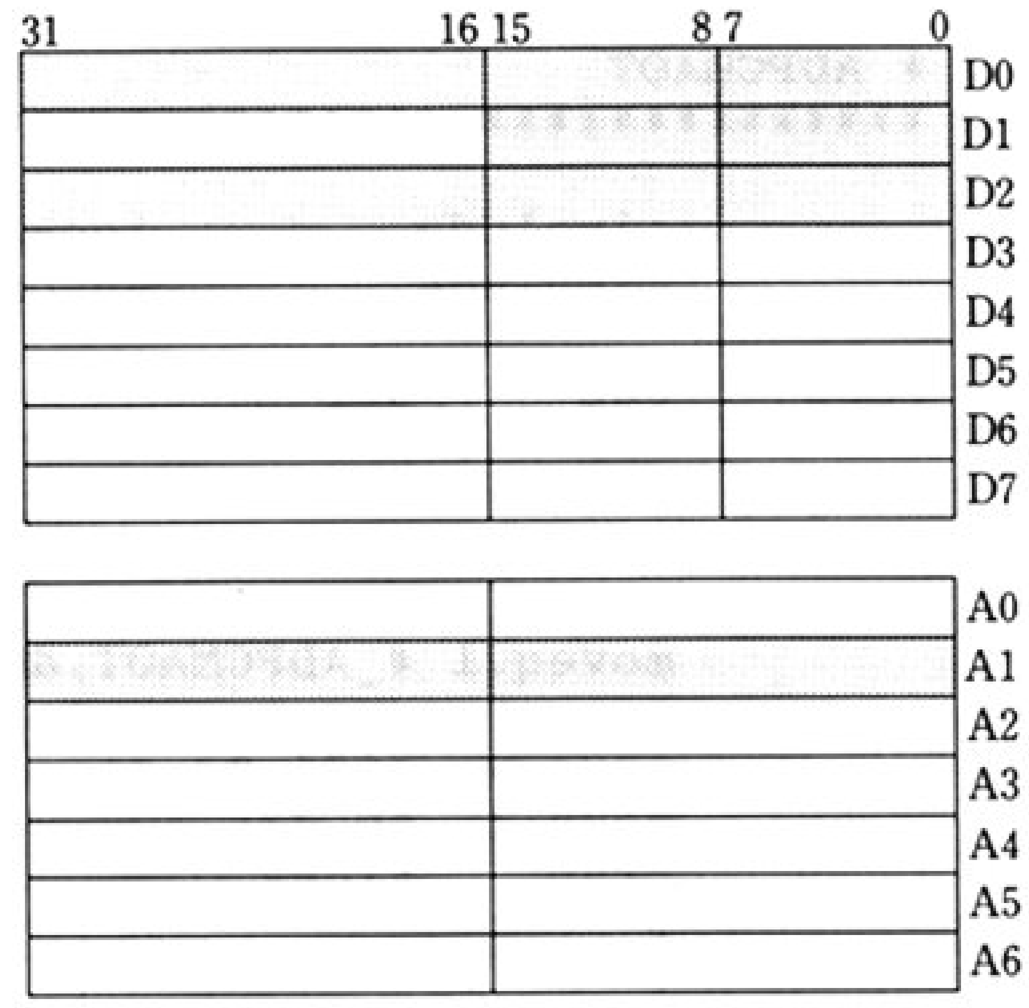
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMINP,d0   * IOCSコール番号をセット
        lea     pcmdat,a1       * データアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        move.l  #7800*10,d2     * 10 秒間 pcmdat に記録されます
        trap    #iocs           * IOCSコール
        .
        .
        .data
pcmdat: ds.b    7800*10         * バッファ 78000 Byte = $130B0 Byte
        .
        .
        end
```

ADPCMAOT

● ADPCMAOT　　IOCS コール番号：$62



機　能　チェーンテーブルによる ADPCM データ出力

入　力

D0. L	$62	IOCS コール番号
A1. L	出力データチェーンテーブルアドレス	
D1. W	サンプリング周波数×256＋出力モード	
D2. L	出力データチェーンテーブルの個数	

リターン　なし（D0. L の内容は保証されません）

解　説　チェーンテーブルによる ADPCM へデータを出力します。

入力の A1. L は、出力データチェーンテーブルアドレスです。

出力データチェーンテーブルの個数 N は、入力の D2. L によって指定されます。

内容は次の通りです。

0 (A1). L　……先頭アドレス 1
4 (A1). W　……長さ 1
6 (A1). L　……先頭アドレス 2
10 (A1). W……長さ 2
⋮
?? (A1). L　……先頭アドレス N
?? (A1). W　……長さ N

長さは$0001～$FFFF の値です。

また、入力の D1. W については、ADPCMOUT（$60）を参照してください。

プログラム例

```
*******************************************
* ADPCMAOT                             $62
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

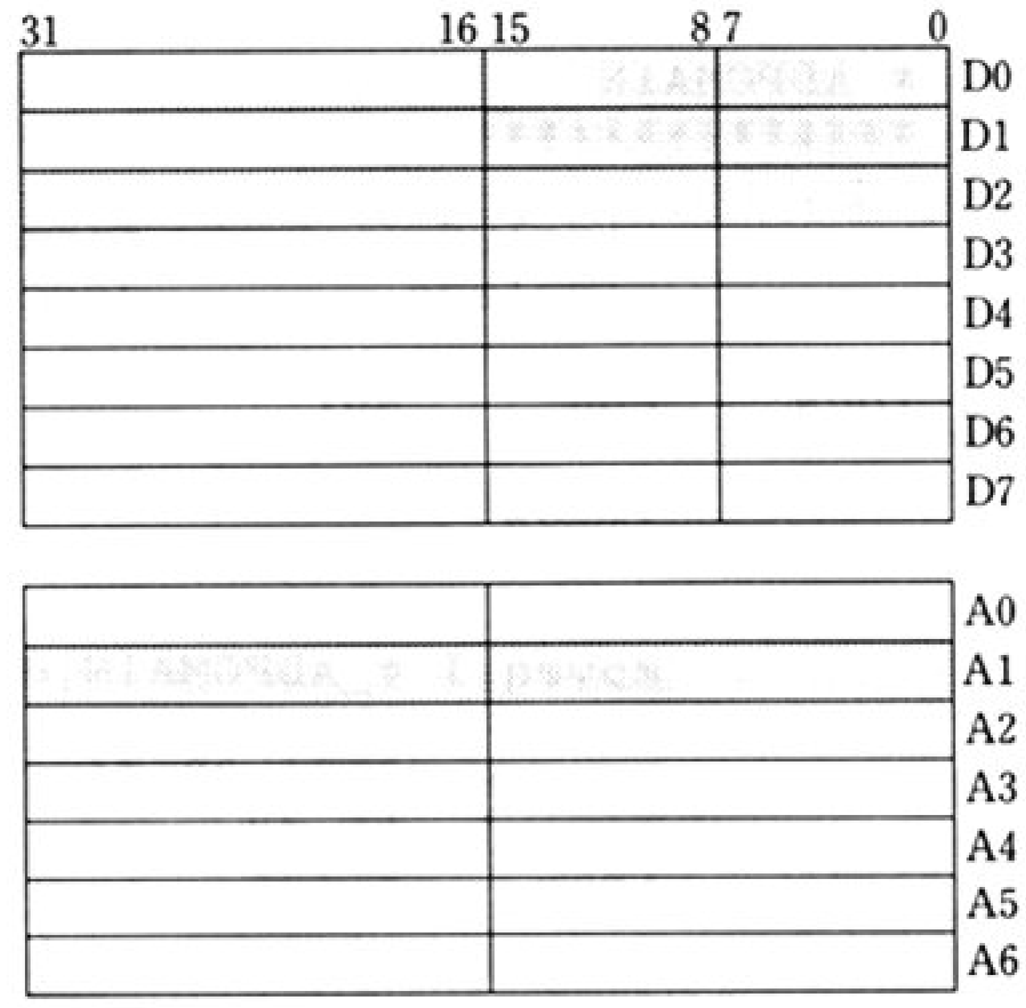
        .text
        .
        .
        moveq.l #_ADPCMAOT,d0    * IOCSコール番号をセット
        lea     pcm_aray,a1      * チェーンアレイテーブルアドレス
        move.w  #$04_03,d1       * 15.6 KHz、左右出力
        move.l  #5,d2            * 5 個のアレイデータを用意してあります
        trap    #iocs            * IOCSコール
        .
        .
        .data
pcm_aray:
        dc.l    aray0
        dc.w    $f3c0            * $f3c0 = 62400 Byte / Block
        dc.l    aray1
        dc.w    $f3c0            * $f3c0 = 62400 Byte / Block
        dc.l    aray2
        dc.w    $f3c0            * $f3c0 = 62400 Byte / Block
        dc.l    aray3
        dc.w    $f3c0            * $f3c0 = 62400 Byte / Block
        dc.l    aray4
        dc.w    $f3c0            * $f3c0 = 62400 Byte / Block

aray0:  ds.b $f3c0
aray1:  ds.b $f3c0
aray2:  ds.b $f3c0
aray3:  ds.b $f3c0
aray4:  ds.b $f3c0
        .end
```

ADPCMAIN

● ADPCMAIN　　IOCS コール番号：$63

機　能　チェーンテーブルによる ADPCM データ入力



入　力

D0. L　$63　IOCS コール番号
A1. L　入力データチェーンテーブルアドレス
D1. W　サンプリング周波数×256＋出力モード
D2. L　入力データチェーンテーブルの個数

リターン　なし（D0. L の内容は保証されません）

解　説　チェーンテーブルによる ADPCM からデータを入力します。
入力の A1. L は、入力データチェーンテーブルアドレスです。
入力データチェーンテーブルの個数 N は、入力の D2. L によって指定されます。
内容については ADPCMAOT（$62）を参照してください。
また、入力の D1. W については、ADPCMINP（$61）を参照してください。

プログラム例　15.6KHz で 8×5＝40 秒間録音します。

```
*******************************************
* ADPCMAIN                            $63
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

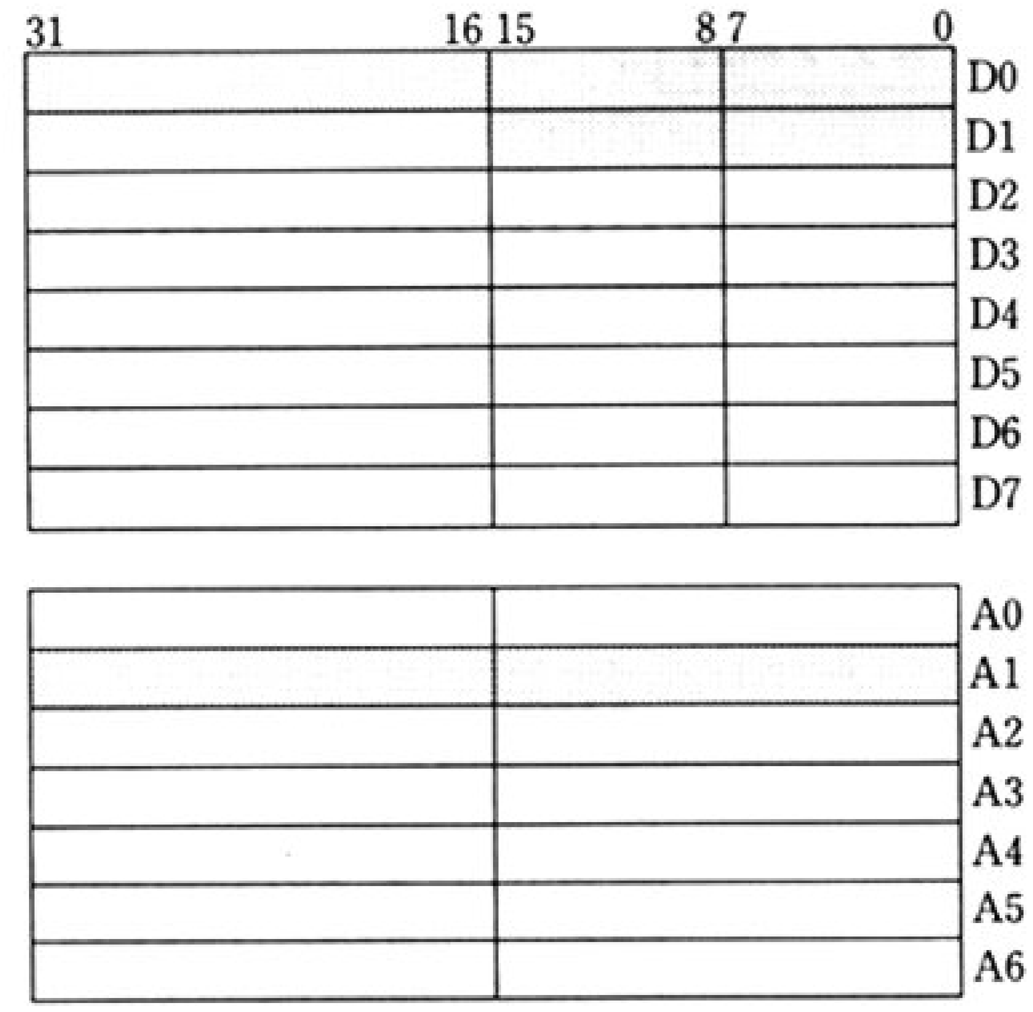
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMAIN,d0   * IOCSコール番号をセット
        lea     pcm_aray,a1     * チェーンアレイテーブルアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        move.l  #5,d2           * 5 個のバッファを用意してあります
        trap    #iocs           * IOCSコール
        .
        .
        .data
pcm_aray:
        dc.l    aray0
        dc.w    $f3c0           * $f3c0 = 62400 Byte / Block
        dc.l    aray1
        dc.w    $f3c0           * $f3c0 = 62400 Byte / Block
        dc.l    aray2
        dc.w    $f3c0           * $f3c0 = 62400 Byte / Block
        dc.l    aray3
        dc.w    $f3c0           * $f3c0 = 62400 Byte / Block
        dc.l    aray4
        dc.w    $f3c0           * $f3c0 = 62400 Byte / Block

aray0:  ds.b $f3c0
aray1:  ds.b $f3c0
aray2:  ds.b $f3c0
aray3:  ds.b $f3c0
aray4:  ds.b $f3c0
        .end
```

ADPCMLOT

● ADPCMLOT　　IOCSコール番号：$64



機　能　アレイチェーンテーブルによる ADPCM データ出力

入　力

D0. L	$64　IOCS コール番号
A1. L	出力データアレイチェーンテーブル先頭アドレス
0 (A1). L	データ先頭アドレス 1
4 (A1). W	長さ 1
6 (A1). L	次のテーブルアドレス 1
0 (?1). L	先頭アドレス 2 (?1 はテーブルアドレス 1)
4 (?1). W	長さ 2
6 (?1). L	次のテーブルアドレス 2
⋮	
0 (?(N−1)). L	先頭アドレス N (?(N−1)はテーブルアドレス(N−1))
4 (?(N−1)). W	長さ N
6 (?(N−1)). L	0　テーブル終わり
D1. W	サンプリング周波数×256＋出力モード

リターン　なし (D0. L の内容は保証されません)

解　説　アレイチェーンテーブルによる ADPCM へデータを出力します。

入力の A1. L は、出力データアレイチェーンテーブル先頭アドレスです。

入力の 0 (A1). L は、先頭アドレス 1 です。

入力の 4 (A1). W は、長さ 1 です。

ただし、長さは$0001～$FFFF の値をとります。

入力の 6 (A1). L は、次のテーブルアドレス 1 です。

入力の 0 (?1). L は、先頭アドレス 2 です。

ただし、?1 は、テーブルアドレス 1 です。

入力の 4 (?1). W は、長さ 2 です。

入力の 6 (?1). L は、次のテーブルアドレス 2 です。

入力の 0 (?(N−1)). L は、先頭アドレス N です。

ただし、?(N−1)は、テーブルアドレス(N−1)です。

入力の 4 (?(N−1)). W は、長さ N です。

入力の 6 (?(N−1)). L は、0 でテーブルの終わりを示します。

また、入力の D1. W については、ADPCMOUT ($60) を参照してください。

プログラム例

```
****************************************
* ADPCMLOT                          $64
****************************************

        .include        iocscall.mac
        .include        doscall.mac

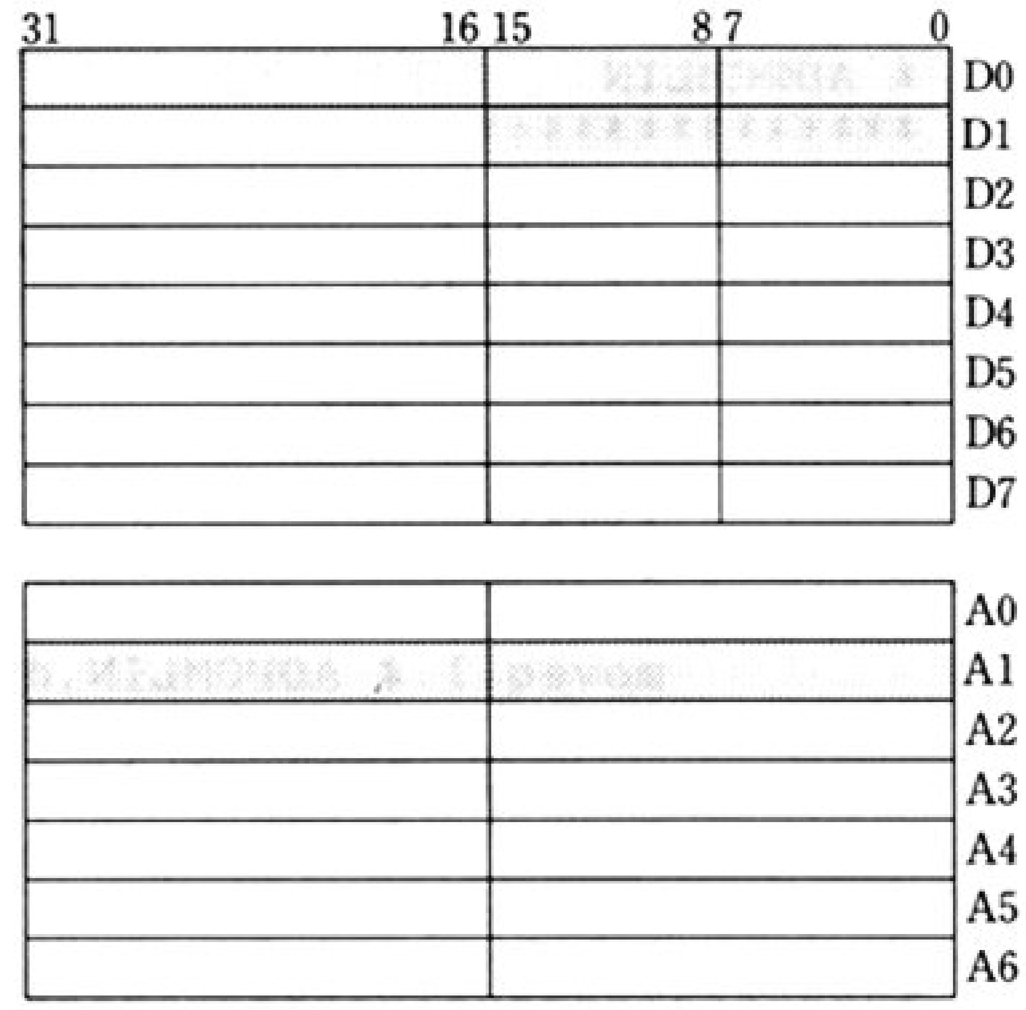
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMLIN,d0   * IOCSコール番号をセット
        lea     aray_tbl,a1     * アレイチェーンテーブルアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        trap    #iocs           * IOCSコール
        .
        .
        moveq.l #_ADPCMLOT,d0   * IOCSコール番号をセット
        lea     aray_tbl,a1     * アレイチェーンテーブルアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        trap    #iocs           * IOCSコール
        .
        .
*----------------------------
** アレイチェーンテーブル
*----------------------------
        .data
aray_tbl:
tbl1:
        dc.l    pcm1            * データバッファ 1 先頭アドレス
        dc.w    $f3c0           * データバッファ 1 のサイズ
        dc.l    tbl2            * 次のテーブルの先頭アドレス
        .
        .
tbl2:
        dc.l    pcm2            * データバッファ 2 先頭アドレス
        dc.w    $f3c0           * データバッファ 2 のサイズ
        dc.l    0               * テーブルの終わりを 0 で表します
        .
        .
*----------------------------
** データバッファ
*----------------------------
pcm1:   ds.b    $f3c0           * データバッファ 1
pcm2:   ds.b    $f3c0           * データバッファ 2
        .
        .
        .end
```

ADPCMLIN

● ADPCMLIN　　IOCS コール番号：$65

機　能　アレイチェーンテーブルによる ADPCM データ入力



入　力

D0. L	$65	IOCS コール番号
A1. L	入力データアレイチェーンテーブル先頭アドレス	
D1. W	サンプリング周波数×256＋出力モード	

リターン　なし（D0. L の内容は保証されません）

解　説　アレイチェーンテーブルによる ADPCM からデータを入力します。
入力の A1. L は、入力データアレイチェーンテーブル先頭アドレスです。
内容については、ADPCMLOT（$64）を参照してください。
また、入力の D1. W については、ADPCMINP（$61；ADPCM からデータを入力）を参照してください。

プログラム例

```
******************************************
* ADPCMLIN                           $65
******************************************

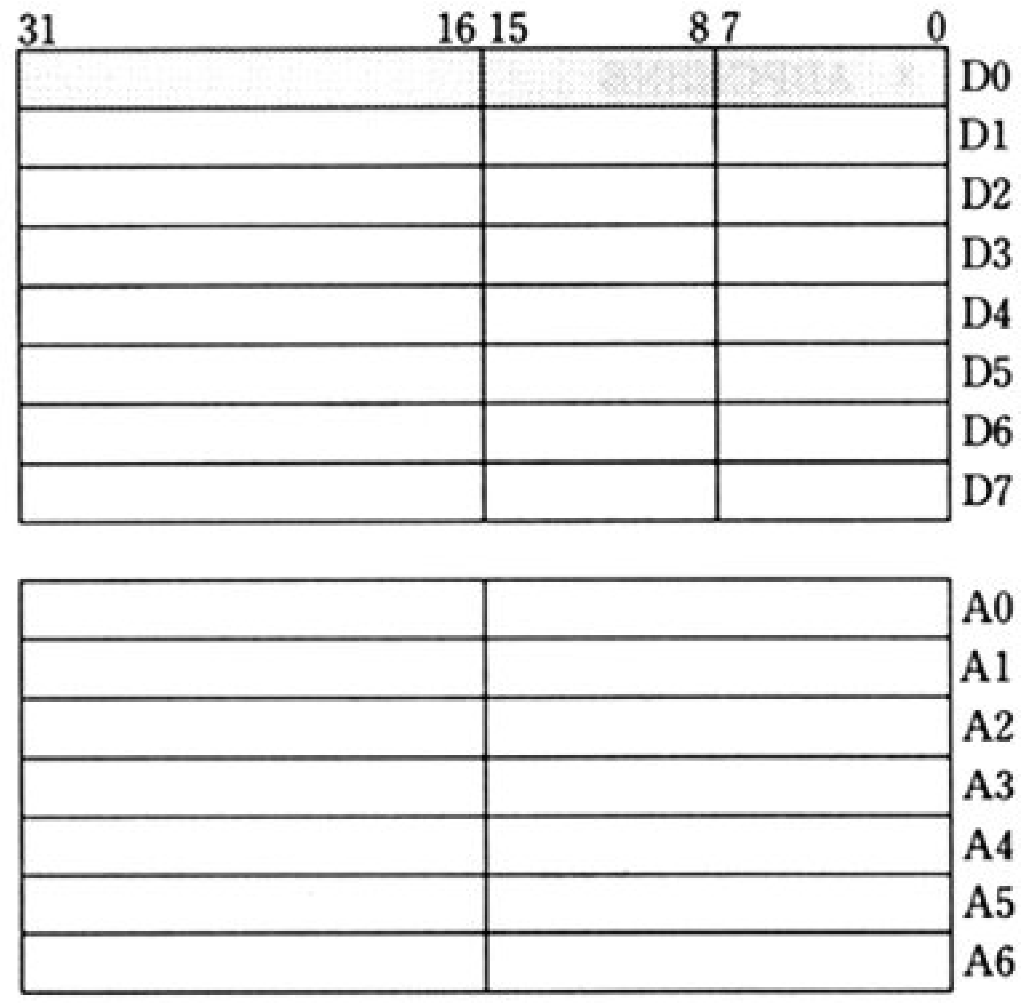
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMLIN,d0   * IOCSコール番号をセット
        lea     aray_tbl,a1     * アレイチェーンテーブルアドレス
        move.w  #$04_03,d1      * 15.6 KHz、左右出力
        trap    #iocs           * IOCSコール
        .
        .
*----------------------------
** アレイチェーンテーブル
*----------------------------
        .data
aray_tbl:
tbl1:
        dc.l    pcm1            * データバッファ 1 先頭アドレス
        dc.w    $f3c0           * データバッファ 1 のサイズ
        dc.l    tbl2            * 次のテーブルの先頭アドレス
        .
        .
tbl2:
        dc.l    pcm2            * データバッファ 2 先頭アドレス
        dc.w    $f3c0           * データバッファ 2 のサイズ
        dc.l    0               * テーブルの終わりを 0 で表します
        .
        .
*----------------------------
** データバッファ
*----------------------------
pcm1:   ds.b    $f3c0           * データバッファ 1
pcm2:   ds.b    $f3c0           * データバッファ 2
        .
        .
        .end
```

ADPCMSNS

● ADPCMSNS　　IOCS コール番号：$66



機　　能　ADPCM の実行モードのセンス

入　　力　D0. L　　$66　　IOCS コール番号

リターン　D0. L　　実行状態

D0. L ＝ $00……何も実行していない
D0. L ＝ $02……出力中
　　ADPCMINP（$60）
　　を実行中
D0. L ＝ $04……入力中
　　ADPCMINP（$61）
　　を実行中
D0. L ＝ $12……出力中
　　ADPCMINP（$62）を実行中
D0. L ＝ $14……入力中
　　ADPCMINP（$63）を実行中
D0. L ＝ $22……出力中
　　ADPCMINP（$64）を実行中
D0. L ＝ $24……入力中
　　ADPCMINP（$65）を実行中

解　　説　ADPCM の実行モードをセンスします。

プログラム例

```
*******************************************
* ADPCMSNS                              $66
*******************************************

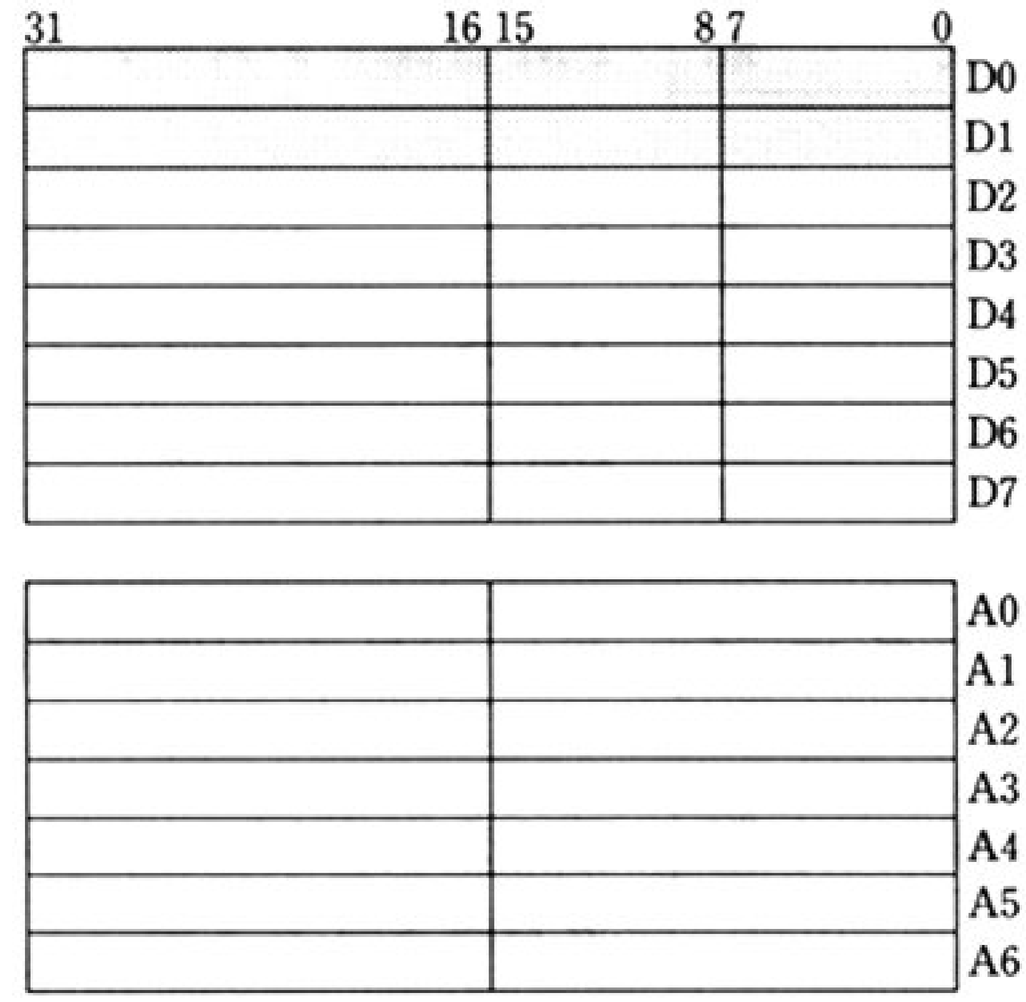
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMSNS,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

ADPCMMOD

● ADPCMMOD　　　IOCS コール番号：$67



機　能　ADPCM の実行制御

入　力　D0. L　$67　IOCS コール番号
D1. L　制御番号

リターン　なし（D0. L の内容は保証されません）

解　説　ADPCM の実行を制御します。
入力の D1. L は制御番号です。
内容は次の通りです。

D0. L ＝ 0……終了
D0. L ＝ 1……中断
D0. L ＝ 2……再開

プログラム例

```
*****************************************
* ADPCMMOD                            $67
*****************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ADPCMMOD,d0    * IOCSコール番号をセット
        moveq.l #0,d1            * 中断します
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

OPMSET

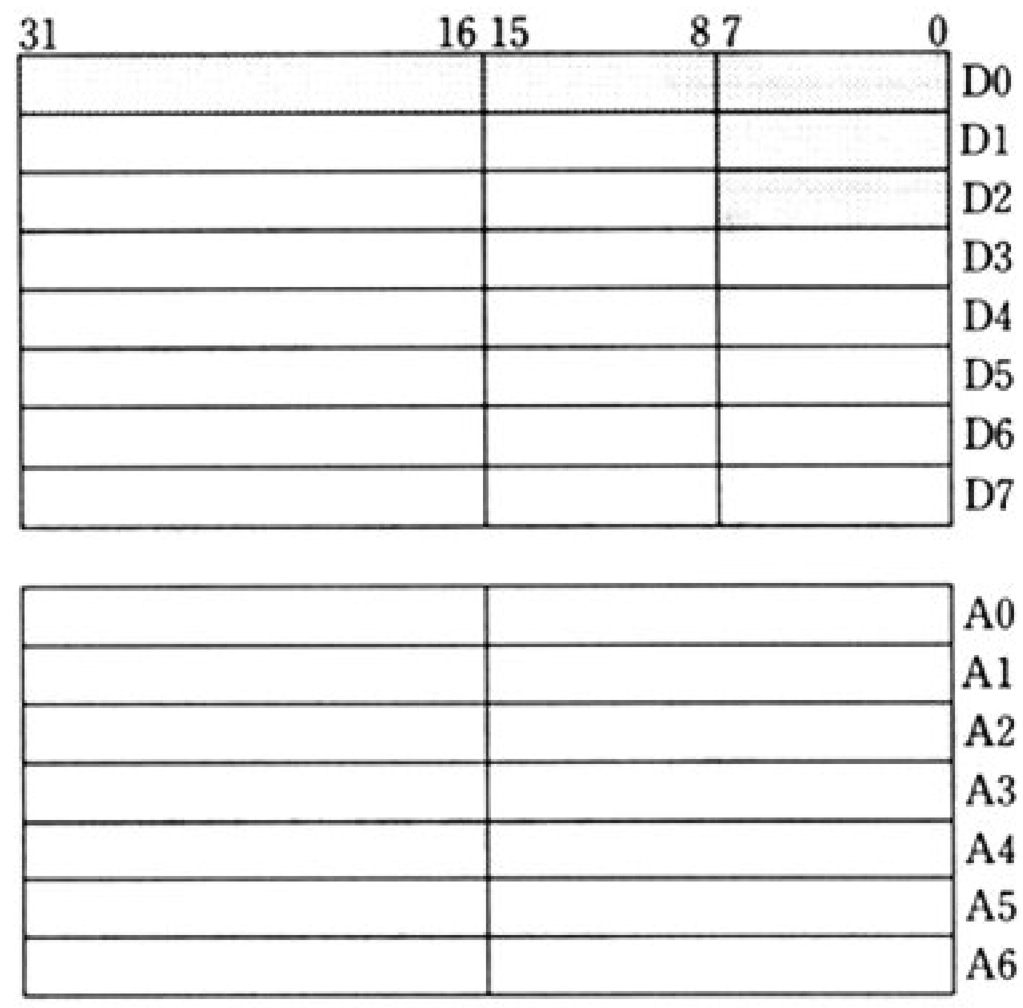
● OPMSET　　IOCS コール番号：$68

機　　能　FM 音源レジスタデータの書き込み

入　　力　D0. L　$68　IOCS コール番号
D1. B　レジスタアドレス
D2. B　データ

リターン　なし（D0. L の内容は保証されません）

解　　説　FM 音源(YM2151)のレジスタに、データを書き込みます。
内部で書き込み可能か判断して書き込みます。
入力の D1. B は、レジスタアドレスです。
入力の D2. B はデータです。



プログラム例

```
*******************************************
* OPMSET                               $68
*******************************************

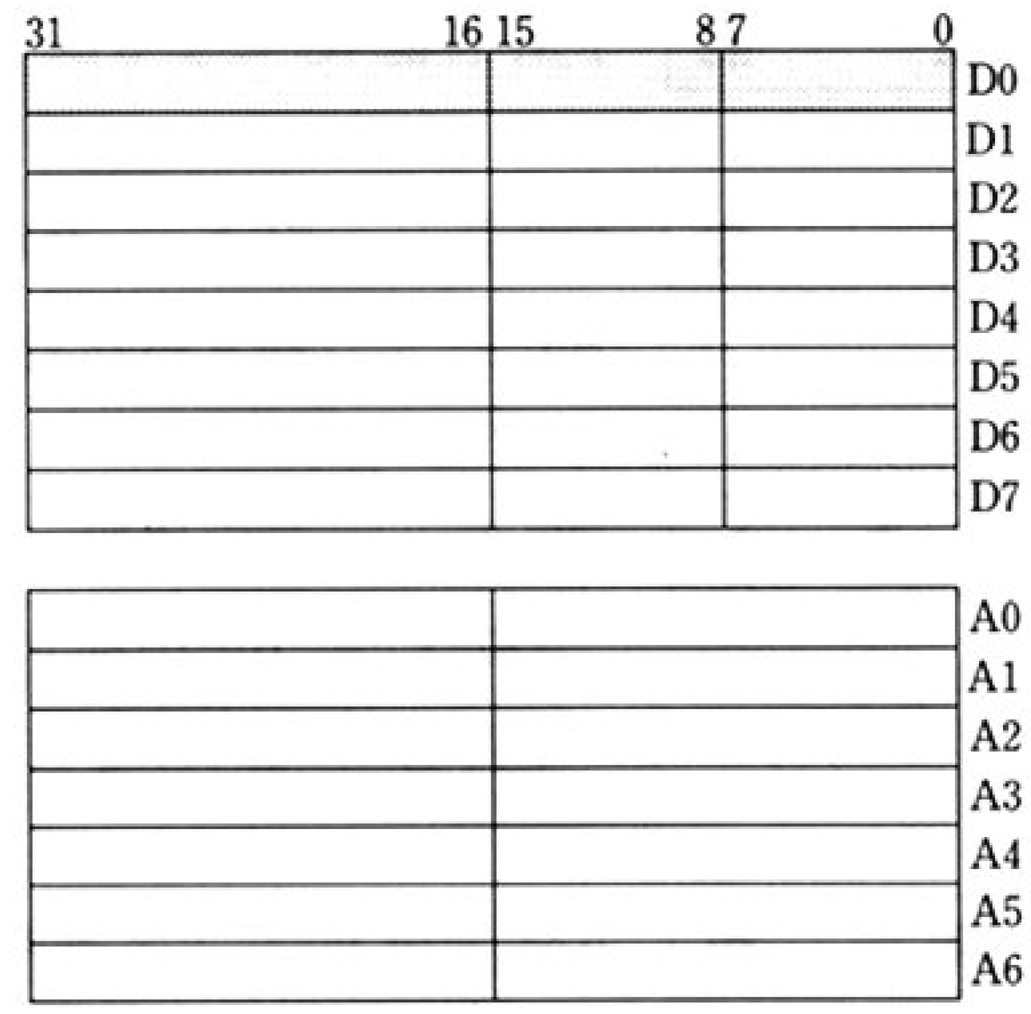
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OPMSET,d0     * IOCSコール番号をセット
        move.b  #$rr,d1         * $rr はレジスタアドレス
        move.b  #$xx,d2         * $xx はデータ
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

OPMSNS

● OPMSNS　　　IOCS コール番号：$69



機能　FM 音源ステータス読み込み

入力　D0. L　　$69　IOCS コール番号

リターン　D0. B　　ステータス

解説　FM 音源(YM2151)のステータスを読み込みます。
リターンの D0. B の内容は、次の通りです。

ビット 7：書き込み禁止フラグ（0 ならばデータ書き込み可)
ビット 1：タイマー A オーバーフロー（1 ならばオーバーフロー)
ビット 0：タイマー B オーバーフロー（1 ならばオーバーフロー)

プログラム例

```
*******************************************
* OPMSNS                               $69
*******************************************

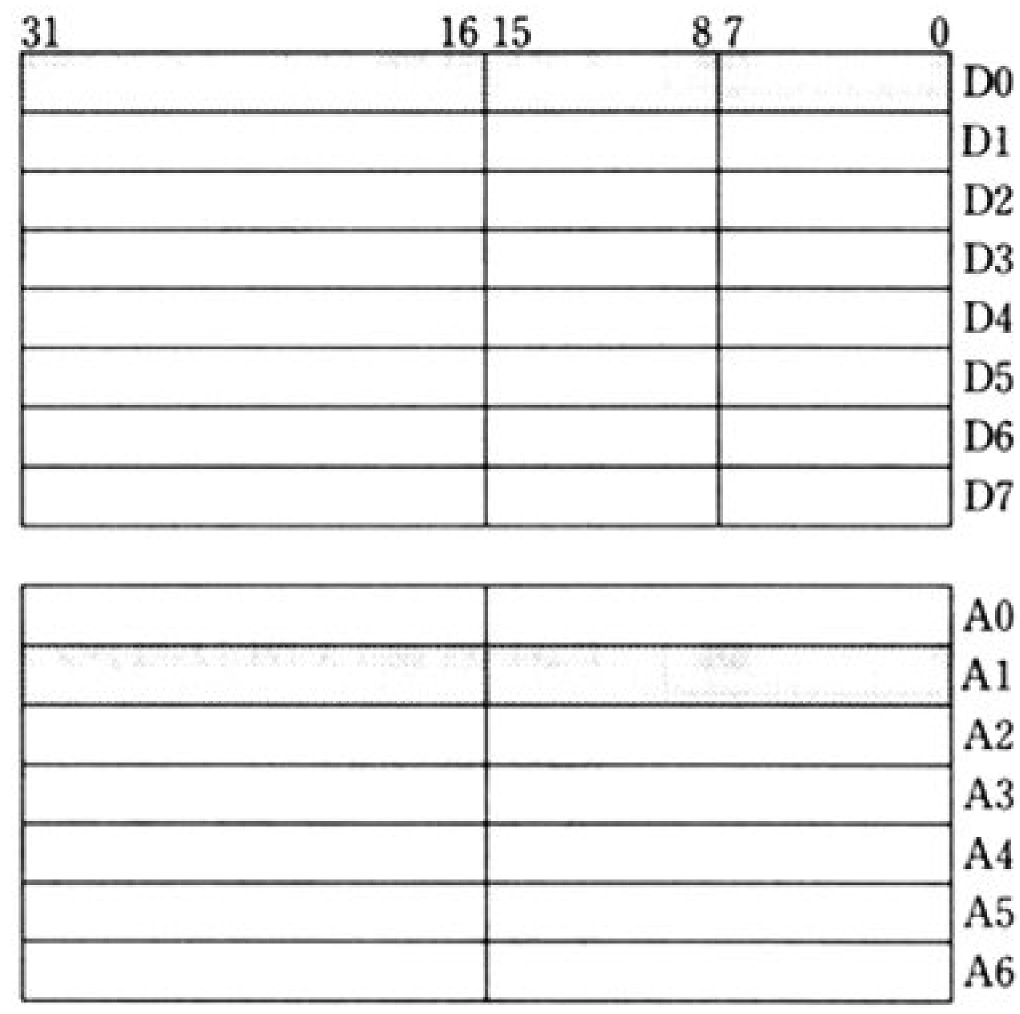
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OPMSNS,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .                       * d0.b にステータスが返ってきます
        .
        .end
```

OPMINTST

● OPMINTST　　IOCS コール番号：$6A



機　　能　FM 音源 IC による割り込み設定

入　　力　D0. L　　$6A　　IOCS コール番号
A1. L　　割り込み処理アドレス

リターン　D0. L　　設定情報
D0. L ＝ 0……割り込みが設定された
D0. L ≠ 0…すでに使用中だった

解　　説　FM 音源 IC による割り込みの設定。
MFP（割り込み制御 IC）のみ制御します。
入力の A1. L が 0 の場合、割り込み禁止です。
それ以外の場合は、値は割り込み処理アドレスです。
なお、FM 音源デバイスドライバ（OPMDRV. X. OPMDRV2. X など）を組んでいる場合は、割り込みを変更してはいけません。

プログラム例

```
*******************************************
* OPMINTST                             $6A
*******************************************

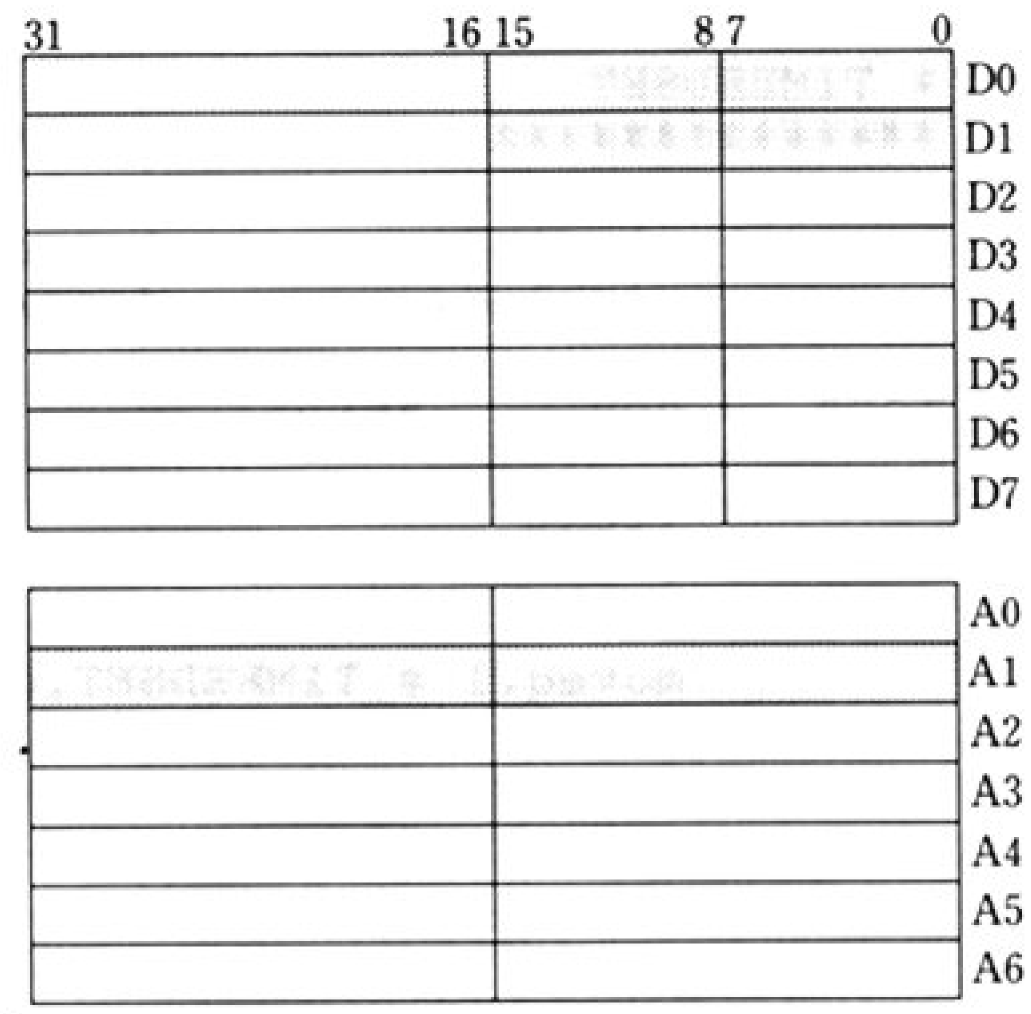
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OPMINTST,d0   * IOCSコール番号をセット
        lea     opmirq,a1       * OPM 割り込み処理アドレス
        trap    #iocs           * IOCSコール
        .
        .
opmirq:                         * 割り込み処理ルーチンを用意してください
        .
        .
        rtr
        .
        .
        .end
```

TIMERDST

● TIMERDST　　IOCS コール番号：$6B



機　　能　MFP の TIMER-D による割り込み

入　　力
D0. L　　$6B　　IOCS コール番号
A1. L　　割り込み処理アドレス
D1. W　　カウンタの単位時間

リターン　D0. L　　設定情報
D0. L ＝ 0……割り込みが設定された
D0. L ≠ 0……すでに使用中

解　　説　MFP の TIMER-D による割り込みの設定を行います。
入力の A1. L が 0 の場合、割り込み禁止です。
それ以外の場合は、値は割り込み処理アドレスです。
入力の D1. W の内容（単位×256＋カウンタ）は次の通りです。

ビット 15～8：単位
カウンタの単位時間（μ 秒単位）を指定します。
内容は次の通りです。
0……指定不可
1……単位時間 1μS
2……単位時間 2.5μS
3……単位時間 4μS
4……単位時間 12.5μS
5……単位時間 16μS
6……単位時間 25μS
7……単位時間 50μS

ビット 7～0　：カウンタ
割り込み周期は単位時間×カウンタになります。
0 は 256 として扱います。
なお、Human68k Ver.2.0 の動作中は、割り込みを変更できません。

プログラム例

```
*******************************************
* TIMERDSET                             $6B
*******************************************

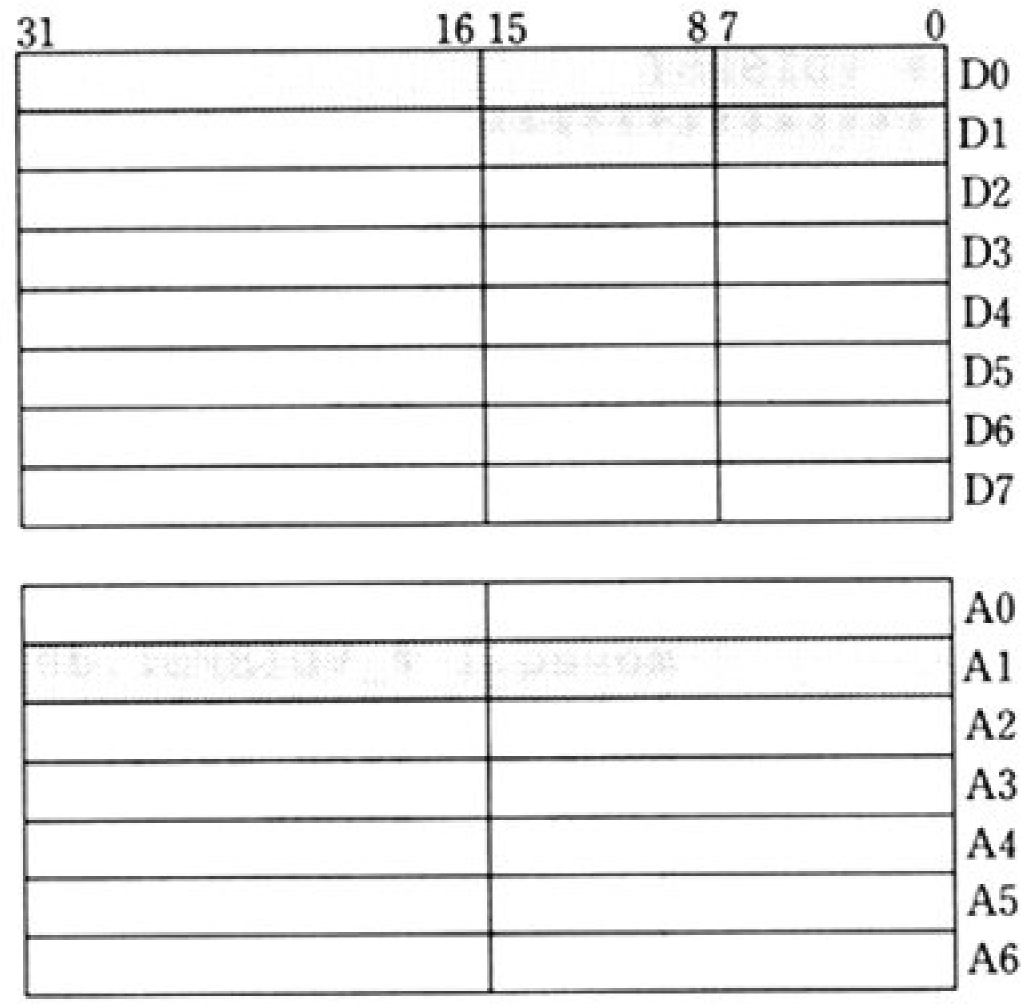
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_TIMERDSET,d0  * IOCSコール番号をセット
        lea     timerd,a1       * タイマー D 割り込み処理アドレス
        move.w  #$01_07         * 7μ秒ごとに割り込み発生
        trap    #iocs           * IOCSコール
        .
        .
timerd:                         * 割り込み処理ルーチンを用意してください
        .
        .
        rte
        .
        .
        .end
```

VDISPST

● VDISPST　　IOCSコール番号：$6C



機　能　垂直周期による割り込み

入　力　D0. L　$6C　IOCSコール番号
A1. L　割り込み処理アドレス
D1. W　期間×256＋カウンタ

リターン　D0. L　設定情報
D0. L ＝ 0……割り込みが設定された
D0. L ≠ 0……すでに使用中

解　説　垂直同期による割り込み。
入力のA1. Lが0の場合、割り込み禁止です。
それ以外の場合は、値は割り込み処理アドレスです。
入力のD1. Wの内容は次の通りです。

ビット15〜8：期間
$00……垂直帰線期間で割り込み
$01……垂直表示期間で割り込み

ビット7〜0 ……カウンタ
割り込み周期はカウンタ×垂直同期周期になります。
0は256として扱います。

プログラム例

```
*******************************************
* VDISPST                               $6C
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

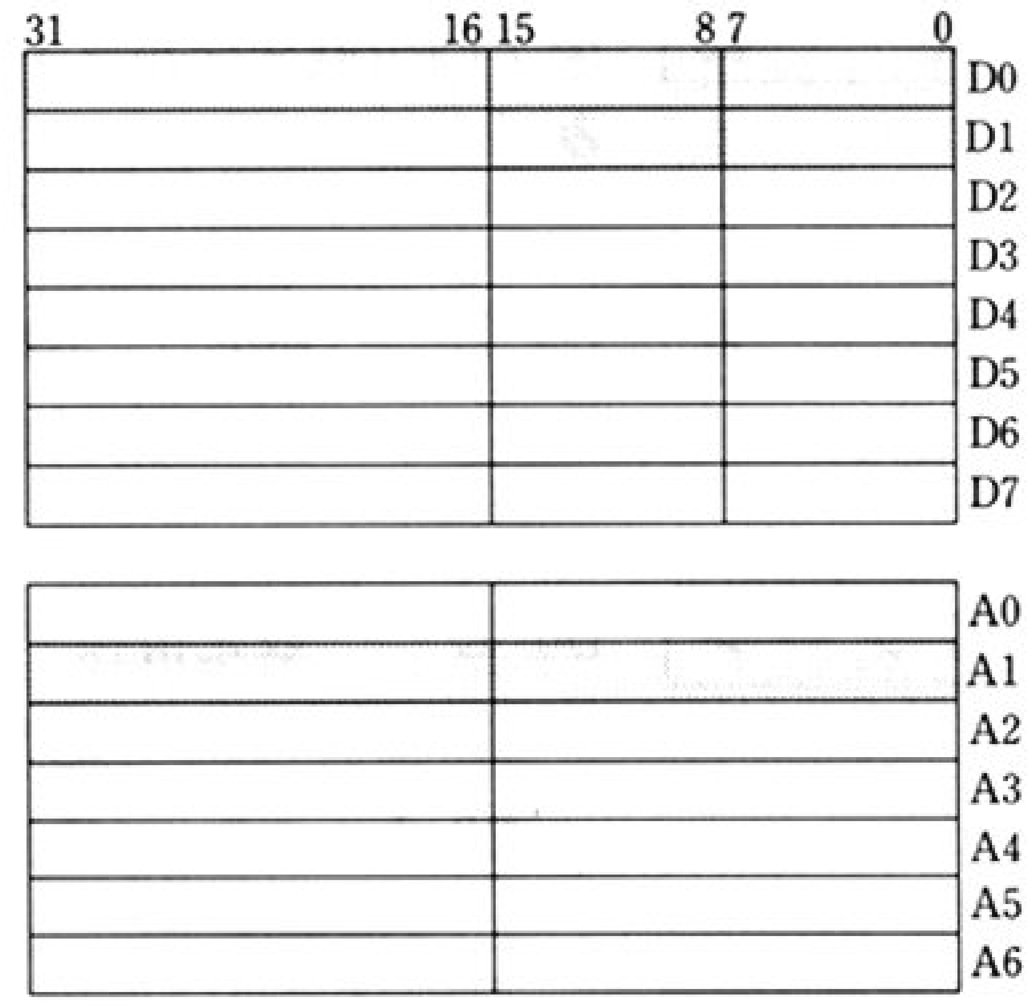
        .text
        .
        .
        moveq.l #_VDISPST,d0     * IOCSコール番号をセット
        lea     vdisp,a1         * 垂直同期 割り込み処理アドレス
        move.w  #$00_07,d1       * 垂直同期 7 回ごとに帰線期間で割り込み
        trap    #iocs            * IOCSコール
        .
        .

vdisp:                           * 割り込み処理ルーチンを用意してください
        .
        .
        rte
        .
        .
        .end
```

CRTCRAS

● CRTCRAS　　IOCS コール番号：$6D

機　能　ラスター走査による割り込み



入　力

D0. L　　$6D　　IOCS コール番号
A1. L　　割り込み処理アドレス
D1. W　　ラスター

リターン

D0. L　　設定情報
　D0. L　＝0……割り込みが設定された
　D0. L　≠0…すでに使用中

解　説

ラスター走査による割り込みです。
入力の A1. L が 0 の場合、割り込み禁止です。
それ以外の場合は、値は割り込み処理アドレスです。
入力の D1. W はラスターです。
CRTC が指定ラスターを走査開始すると、割り込みが発生します。

プログラム例

```
*******************************************
* CRTCRAS                              $6D
*******************************************

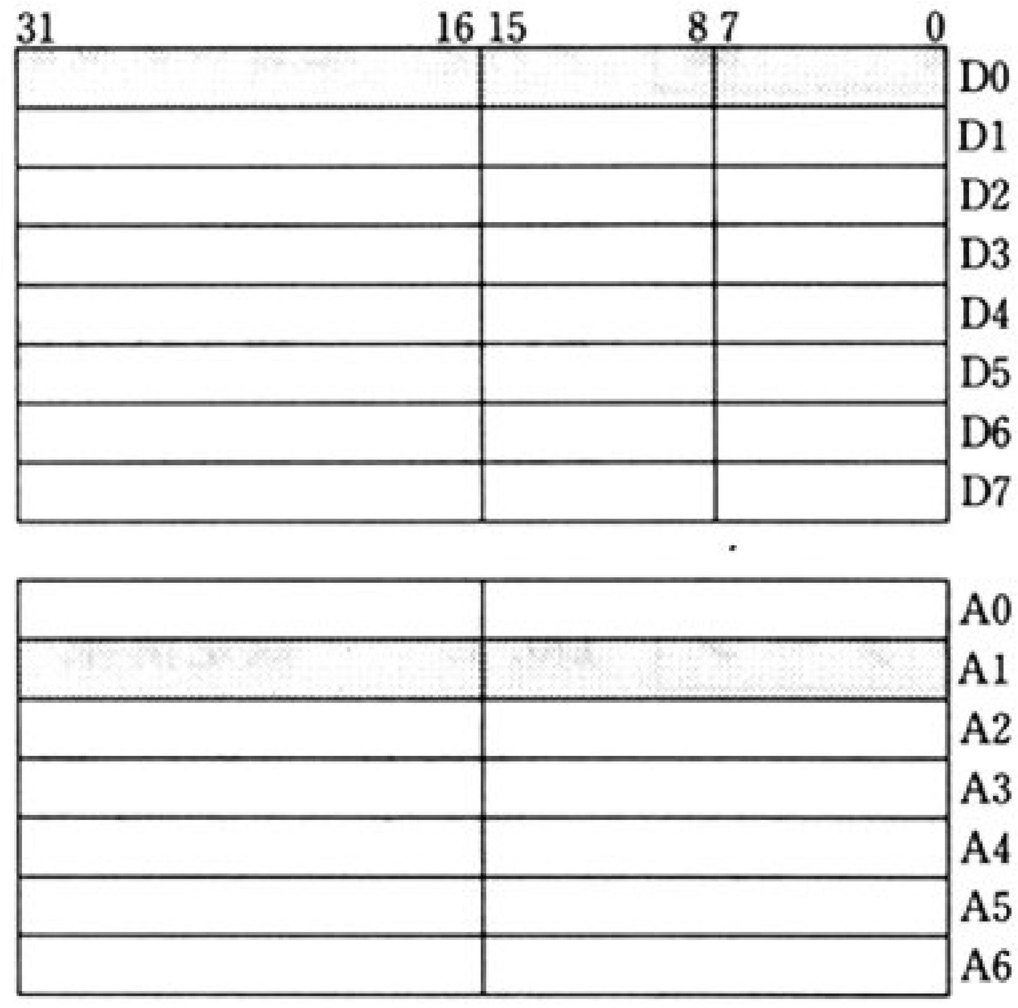
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_CRTCRAS,d0     * IOCSコール番号をセット
        lea     rasirq,a1        * ラスター走査割り込み処理アドレス
        move.w  #$00,d1          * ラスター 0 を指定
        trap    #iocs            * IOCSコール
        .
        .
rasirq:                          * 割り込み処理ルーチンを用意してください
        .
        .
        rte
        .
        .
        .end
```

HSYNCST

● HSYNCST　　IOCS コール番号：$6E



機　　能　水平同期信号の立ち下がりによる割り込み

入　　力　D0. L　$6E　IOCS コール番号
A1. L　割り込み処理アドレス

リターン　D0. L　設定情報
D0. L　=0……割り込みが設定された
D0. L　≠0……すでに使用中

解　　説　水平同期信号の立ち下がりによる割り込みです。
入力の A1. L が 0 の場合、割り込み禁止です。
それ以外の場合は、値は割り込み処理アドレスです。

プログラム例

```
*********************************************
* HSYNCST                                 $6E
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

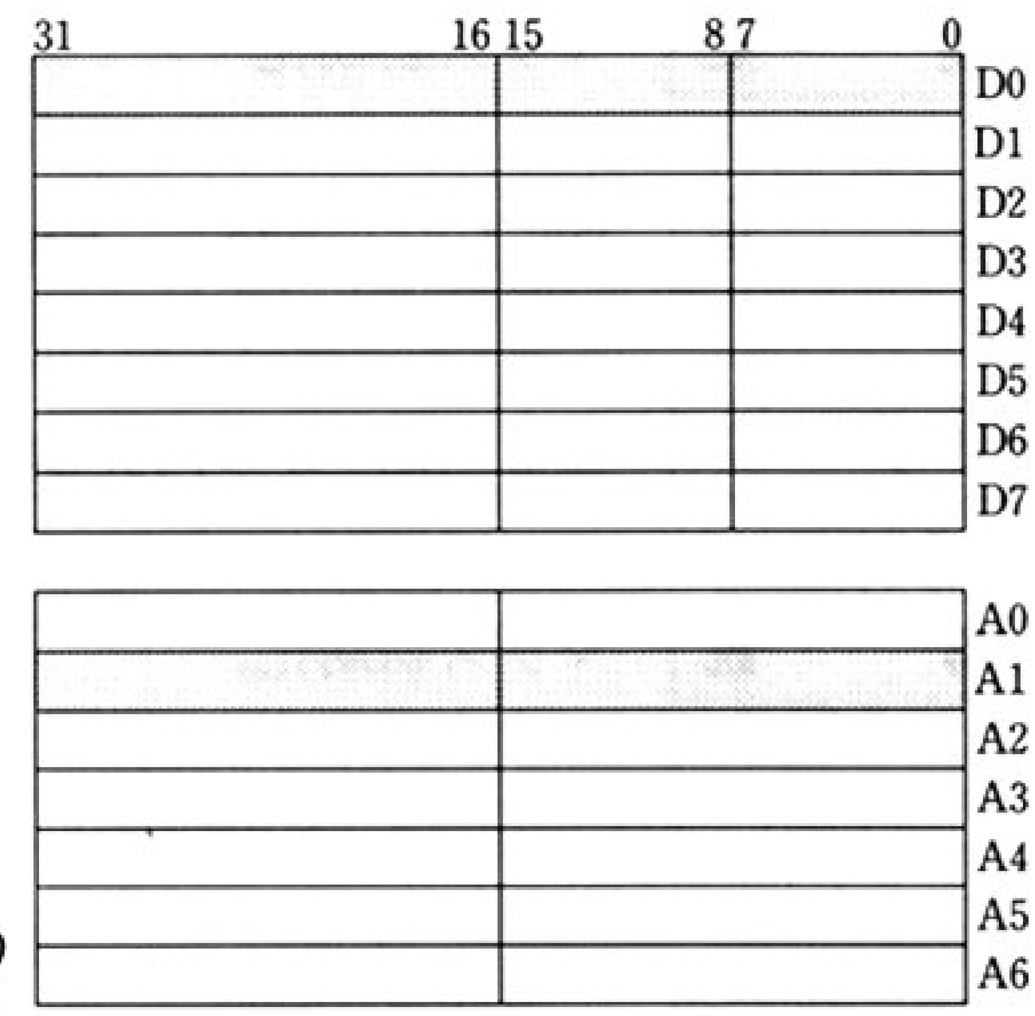
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_HSYNCST,d0    * IOCSコール番号をセット
        lea     sync_irq,a1     * 水平同期割り込み処理アドレス
        trap    #iocs           * IOCSコール
        .
        .

sunc_irq:                       * 割り込み処理ルーチンを用意してください
        .
        .
        rte
        .
        .
        .end
```

PRNINTST

● PRNINTST　　IOCS コール番号：$6F



機　能　プリンタによる割り込み

入　力　D0. L　　$6F　　IOCS コール番号
A1. L　　割り込み処理アドレス

リターン　D0. L　　設定情報
D0. L　＝0……割り込みが設定された
D0. L　≠0……すでに使用中

解　説　PRNINTST は、プリンタが X68000 より送られるデータを受け付けることが可能となった時、発生する割り込みの処理を設定する IOCS コールです。
入力の A1. L が 0 の時、割り込みは禁止になります。
0 でない時、その値が割り込み処理を行うプログラムの先頭アドレスとなります。

プログラム例

```
********************************************
* PRNINTST                              $6F
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_PRNINTST,d0   * IOCSコール番号をセット
        lea     prn_irq,a1      * プリンタ割り込み処理アドレス
        trap    #iocs           * IOCSコール
        .
        .

prn_irq:                        * 割り込み処理ルーチンを用意してください
        .
        .
        rte
        .
        .
        .end
```

MS_INIT

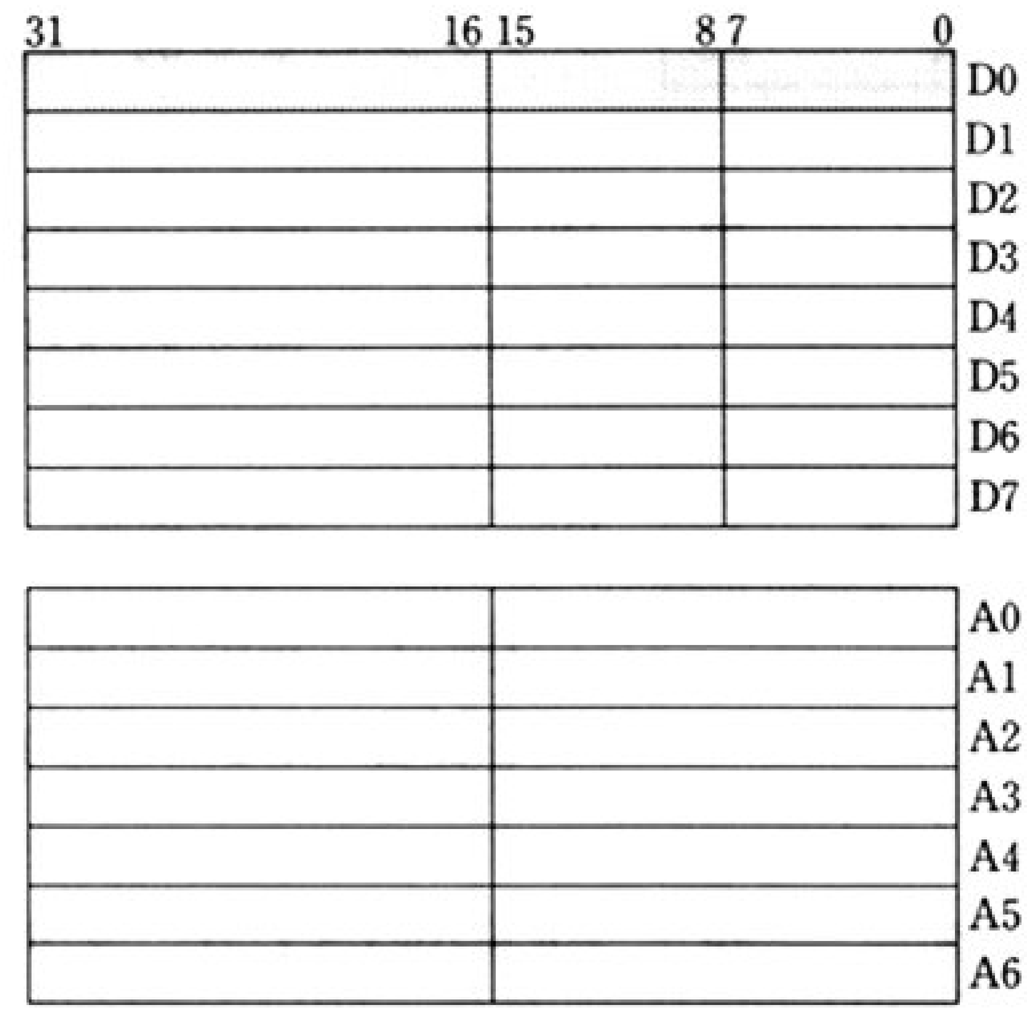
● MS_INIT　　IOCS コール番号：$70

機　　能　マウス初期化

入　　力　D0. L　　$70　　IOCS コール番号

リターン　なし（D0. L の内容は保証されません）

解　　説　マウス初期化。



プログラム例

```
*******************************************
* MS_INIT                              $70
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

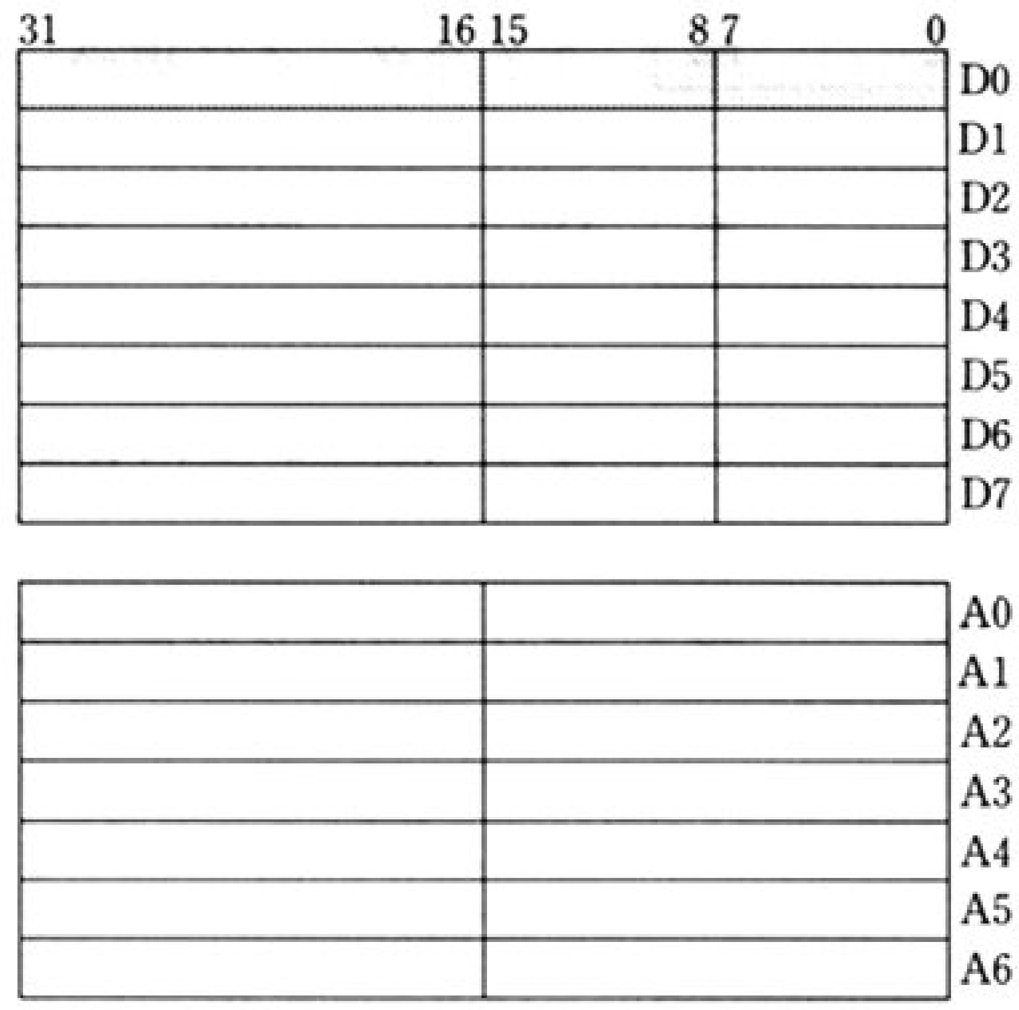
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_INIT,d0     * IOCSコール番号をセット
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

MS_CURON

● MS_CURON　　IOCS コール番号：$71

機　能	マウスカーソル表示
入　力	D0. L　　$71　　IOCS コール番号
リターン	なし（D0. L の内容は保証されません）
解　説	マウスカーソル表示。



プログラム例

```
**********************************************
* MS_CURON                                $71
**********************************************

        .include        iocscall.mac
        .include        doscall.mac

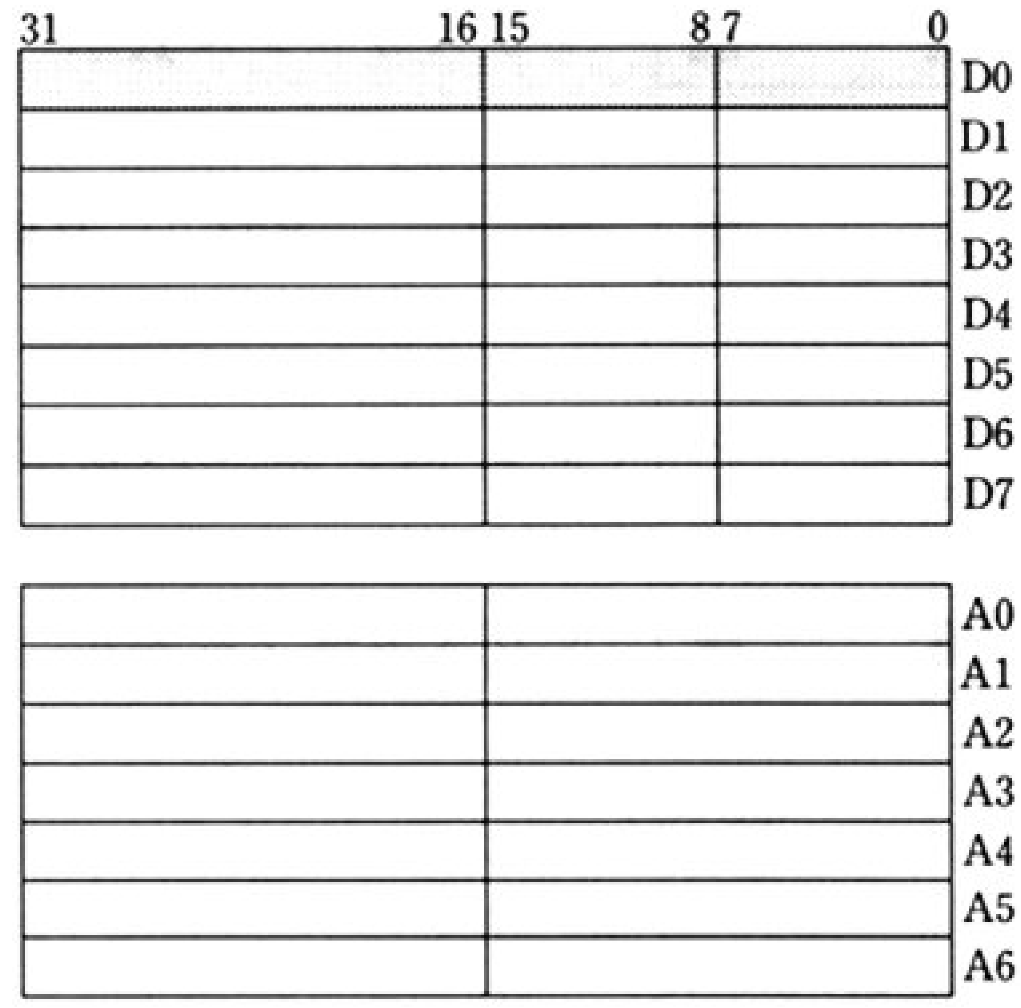
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_CURON,d0     * IOCSコール番号をセット
        trap    #iocs             * IOCSコール
        .
        .
        .end
```

MS_CUROF

● MS_CUROF　　IOCS コール番号：$72

機　能	マウスカーソル消去
入　力	D0. L　$72　IOCS コール番号
リターン	なし（D0. L の内容は保証されません）
解　説	マウスカーソルを消去します。



プログラム例

```
*******************************************
* MS_CUROF                            $72
*******************************************

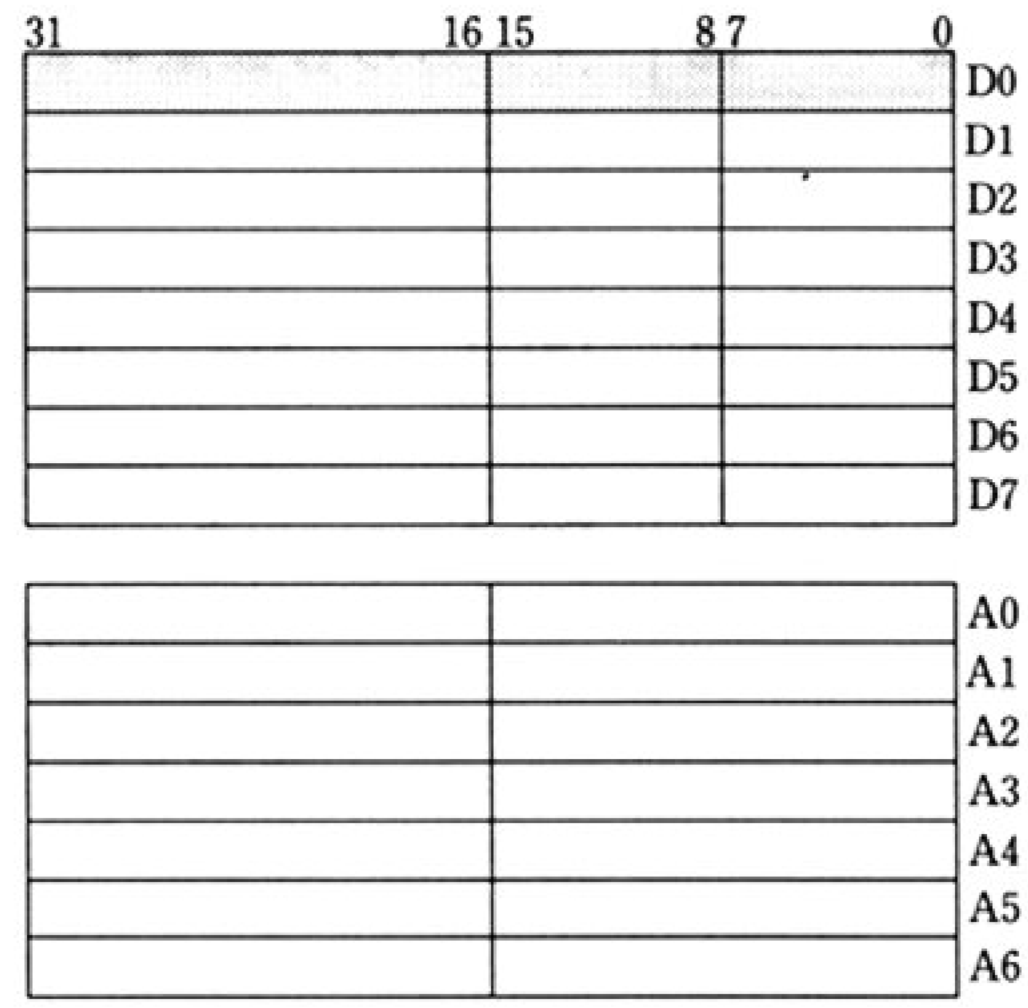
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_CUROF,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

MS_STAT

● MS_STAT　　IOCS コール番号：$73



機　能　マウスカーソルの表示モードチェック

入　力　D0. L　$73　IOCS コール番号

リターン　D0. W　ステータス
　D0. W　＝ 0……マウスカーソルは表示されていない
　D0. W　＝ −1…マウスカーソルは表示されている

解　説　マウスカーソルの表示モードを調べます。

プログラム例

```
*********************************************
* MS_STAT                                $73
*********************************************

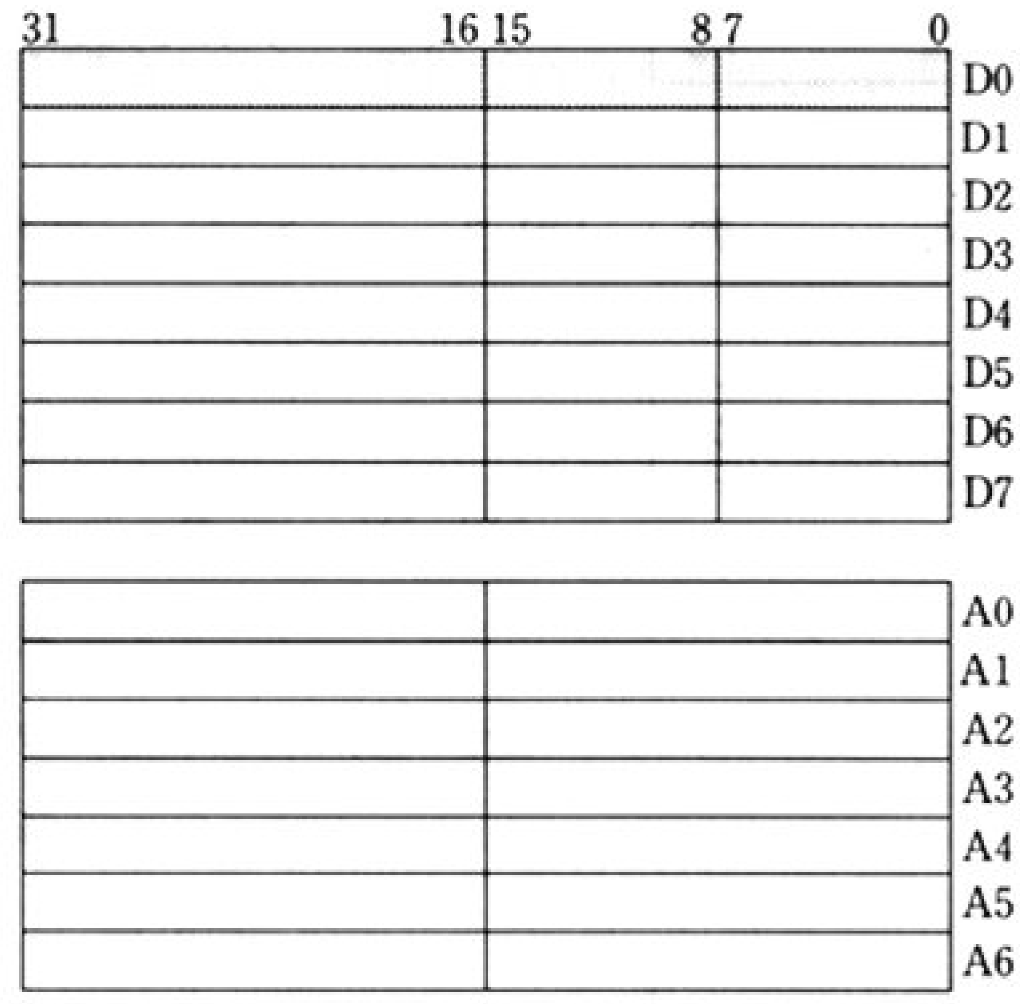
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_STAT,d0      * IOCSコール番号をセット
        trap    #iocs             * IOCSコール
        .
        .
        .end
```

MS_GETDT

● MS_GETDT　　IOCS コール番号：$74



機　　能　マウスの移動量およびボタンのON／OFF チェック

入　　力　D0. L　$74　IOCS コール番号

リターン　D0. L　マウスの移動量、およびボタンのON／OFF の状態を返します。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0.L	X 方向移動量								Y 方向移動量							

	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
	LL								RR							

LL　=$00 ……左ボタン OFF
LL　=$FF……左ボタン ON
RR　=$00 ……右ボタン OFF
RR　=$FF……右ボタン ON

解　　説　マウスの移動量やボタンのON／OFF を調べます。

プログラム例

```
*******************************************
* MS_GETDT                             $74
*******************************************

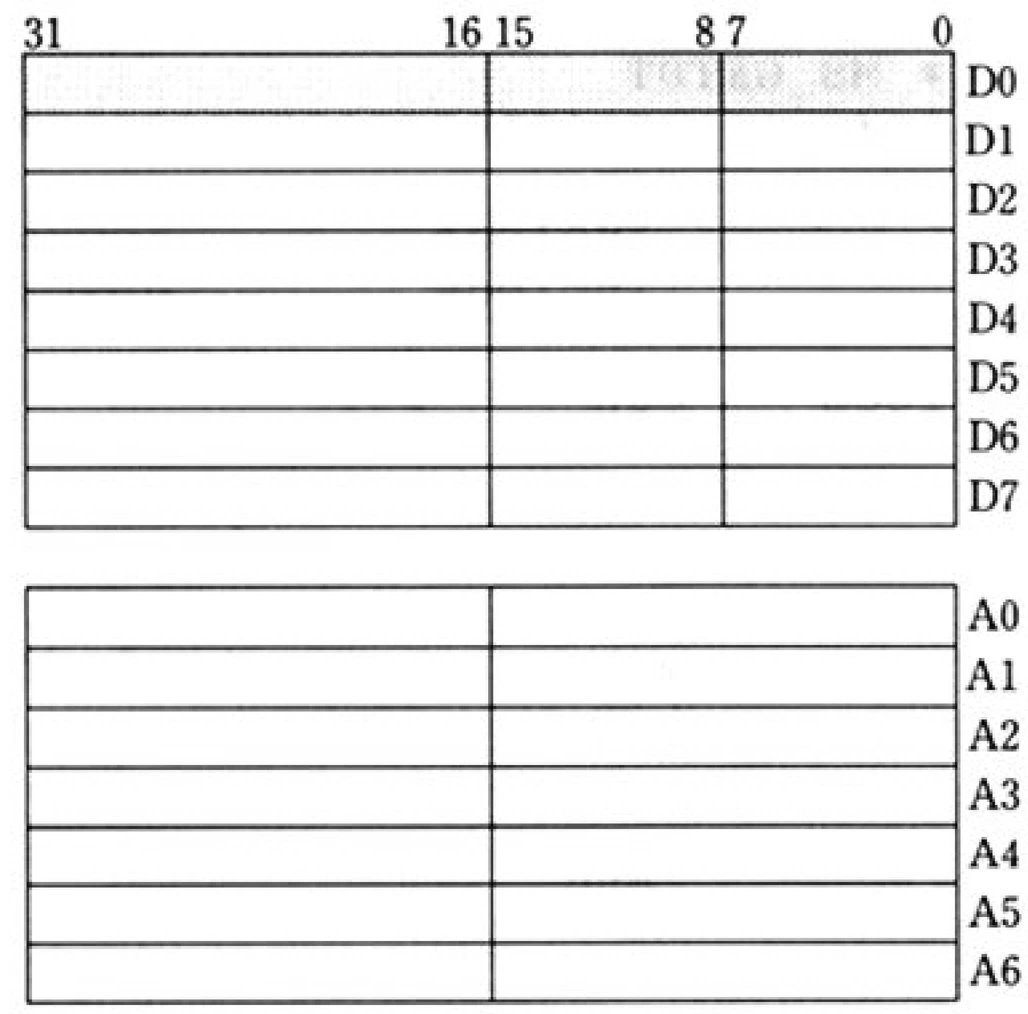
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_GETDT,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

MS_CURGT

● MS_CURGT　　　IOCS コール番号：$75



機　　能　マウスカーソルの座標の取得

入　　力　D0. L　　$75　IOCS コール番号

リターン　D0. L　　マウスカーソルの座標

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0. L	X 座標															
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D09**	**D08**	**D07**	**D06**	**D05**	**D04**	**D03**	**D02**	**D01**	**D00**
	Y 座標															

解　　説　マウスカーソルの座標を調べます。

プログラム例

```
*********************************************
* MS_CURGT                              $75
*********************************************

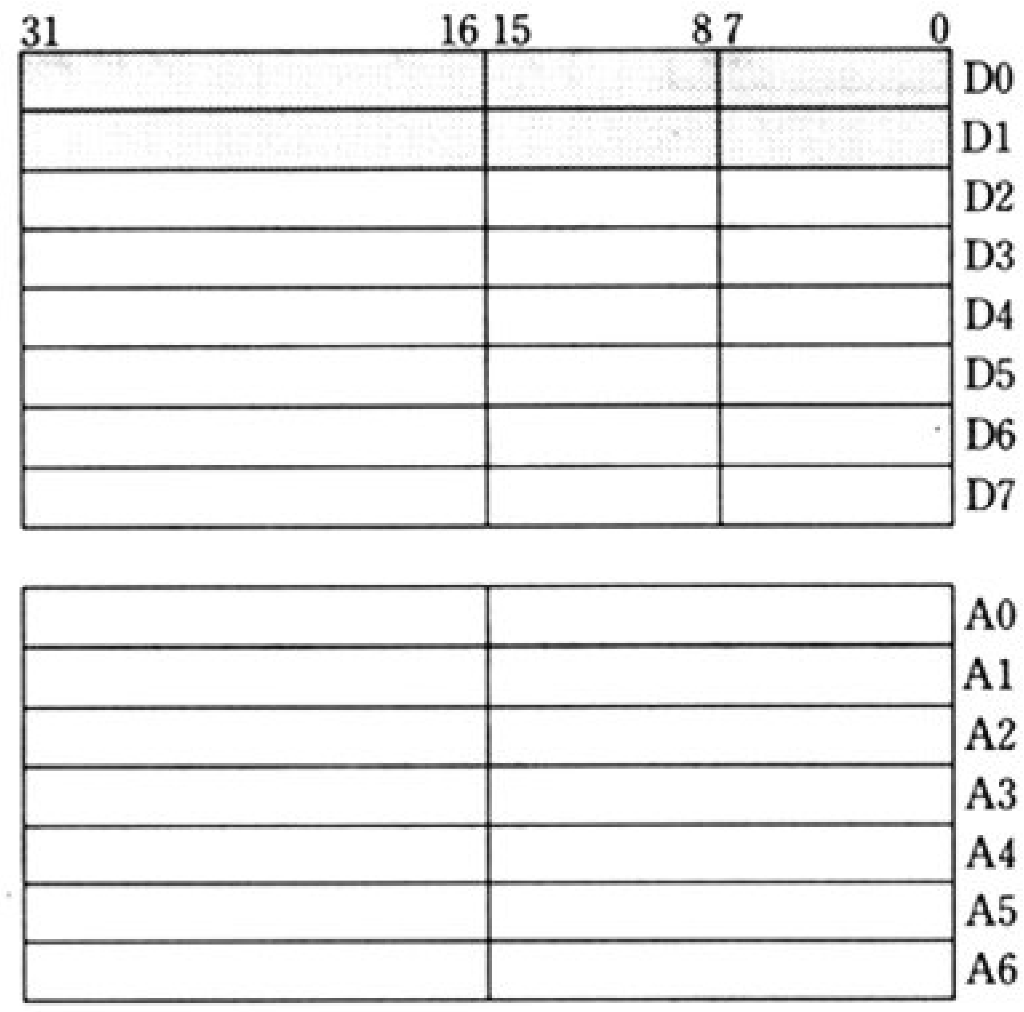
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_CURGT,d0    * IOCSコール番号をセット
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

MS_CURST

● MS_CURST　　IOCSコール番号：$76



機　能　マウスカーソルの座標を指定する

入　力　D0. L　$76　IOCSコール番号
D1. L　マウスカーソルの座標

リターン　D0. L　終了コード
D0. L　＝ 0……座標が指定できた
D0. L　＝ −1…座標が異常

解　説　マウスカーソルの座標を指定します。
入力のD1. Lの内容は次の通りです。

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D1.L	X座標															
	D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D09**	**D08**	**D07**	**D06**	**D05**	**D04**	**D03**	**D02**	**D01**	**D00**
	Y座標															

プログラム例

```
*******************************************
* MS_CURST                             $76
*******************************************

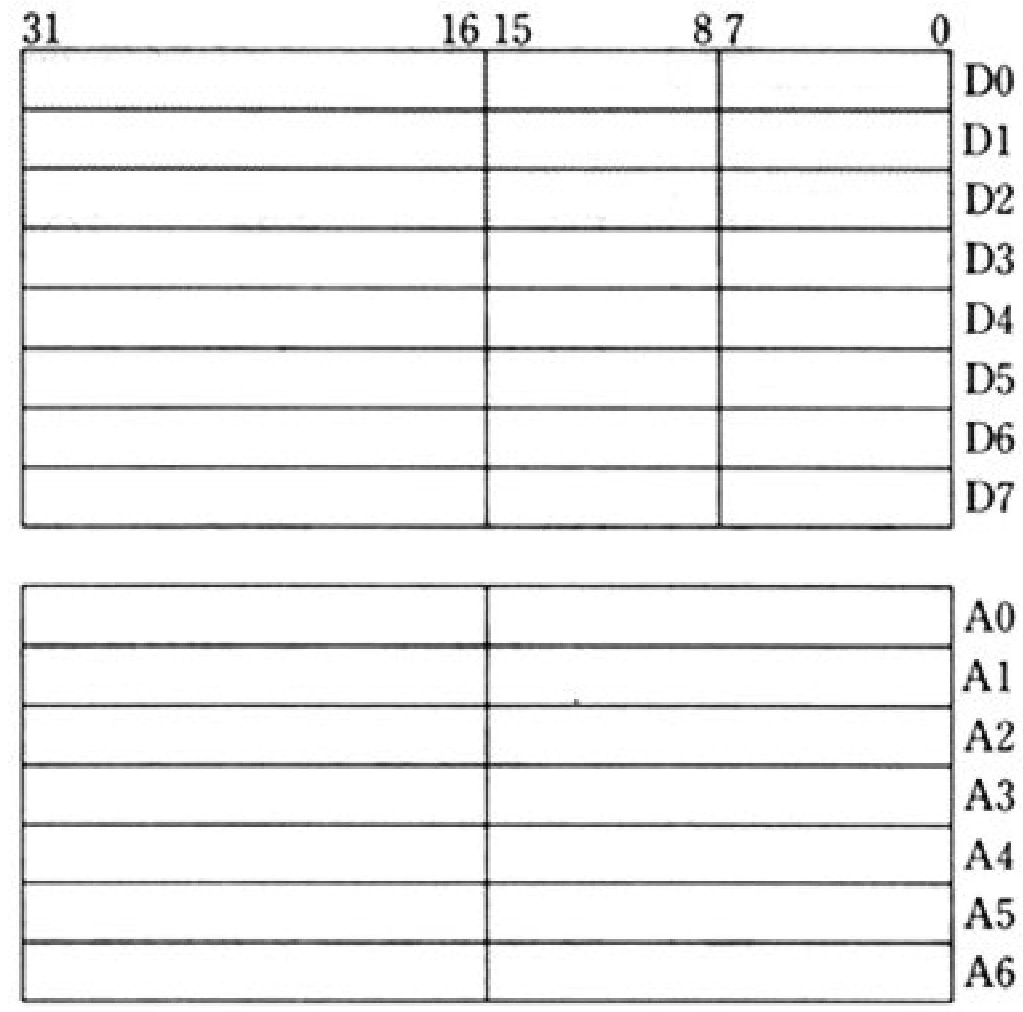
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_CURST,d0   * IOCSコール番号をセット
        move.l  #$0000_0000,d1  * x = 0、y = 0
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

MS_LIMIT

● MS_LIMIT　　IOCS コール番号：$77



機　　能　マウスカーソルの移動範囲指定

入　　力　D0. L　　$77　　IOCS コール番号
D1. L　　先頭座標
D2. L　　最終座標

リターン　D0. L　　終了コード
D0. L　＝ 0……座標が指定できた
D0. L　＝ −1…座標が異常

解　　説　マウスカーソルの移動範囲を指定します。
入力の D1. L の内容は次の通りです。
左上すみを指定します。

D1. L

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
先頭 X 座標															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
先頭 Y 座標															

入力の D2. L の内容は次の通りです。
右下すみを指定します。

D2. L

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
最終 X 座標															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
最終 Y 座標															

プログラム例

```
*******************************************
* MS_LIMIT                             $77
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

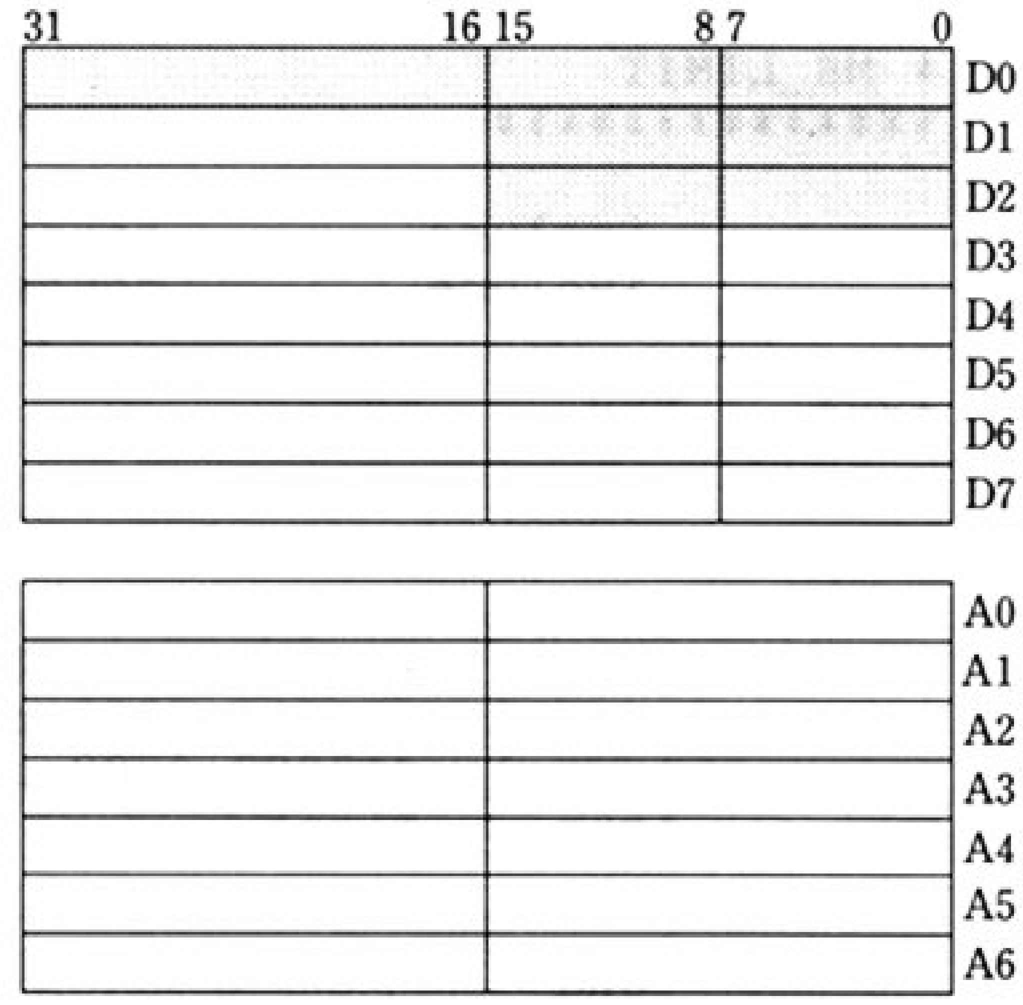
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_LIMIT,d0    * IOCSコール番号をセット
        move.l  #$0000_0000,d1
        move.l  #$0200_0200,d2   * x < $200 (512)、y < $200 (512)
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

MS_OFFTM

● MS_OFFTM　　　IOCS コール番号：$78

機能　マウスボタンを離すまでの時間のチェック



入力　
D0. L　$78　IOCS コール番号
D1. W　ボタンの指定
D2. W　待ち時間

リターン　
D0. W　待ち時間(1～65534)
　D0. W　=0 ……ドラッグ
　D0. W　=−1 …待ち時間の最大値を超えた

解説　
マウスボタンを離すまでの時間を調べます。
入力の D1. W は、ボタンの指定です。
0 ならば左ボタン、−1 ならば右ボタンです。
入力の D2. W は、待ち時間です。
0 ならばいつまでも待ちます。
それ以外の場合は、値が待ち時間の最大値です。

プログラム例

```
******************************************
* MS_OFFTM                            $78
******************************************

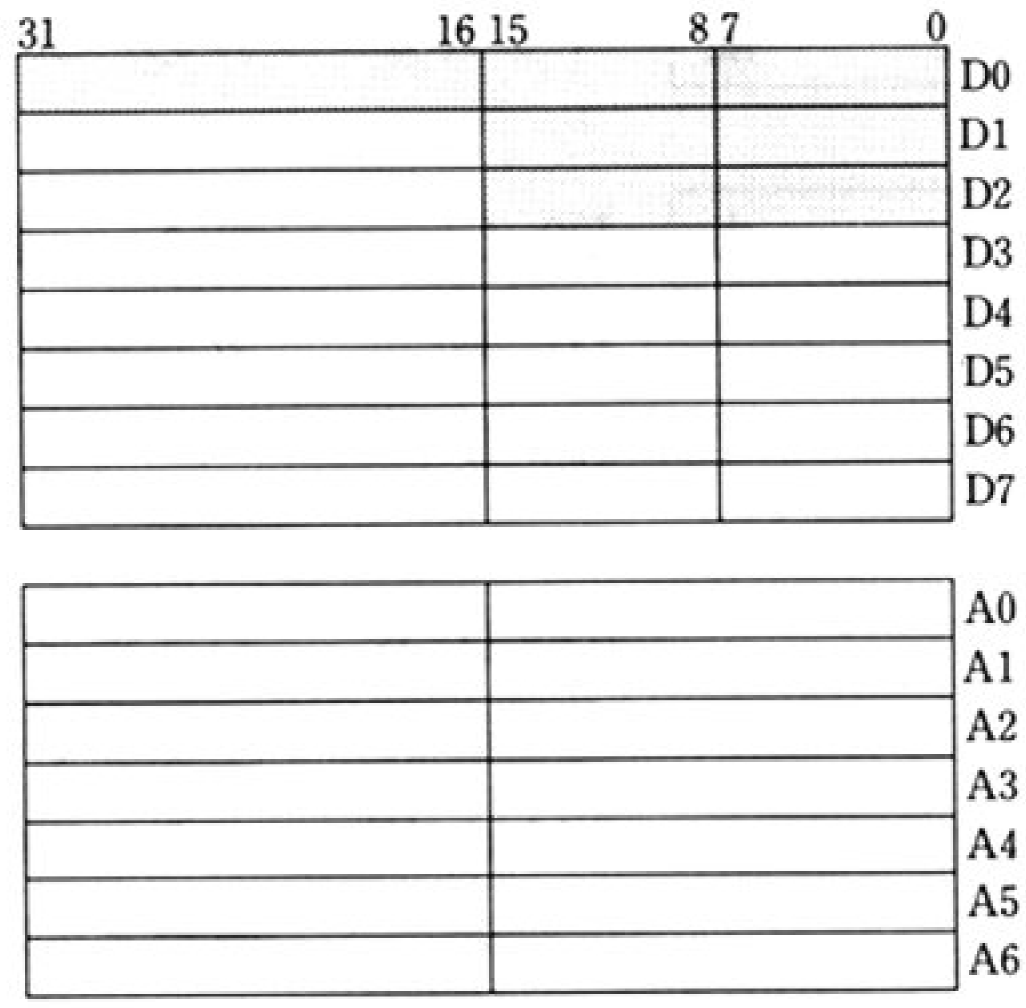
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_OFFTM,d0     * IOCSコール番号をセット
        move.w  #-1,d1            * 右ボタン
        move.w  #1000,d2
        trap    #iocs             * IOCSコール
        .
        .
        .end
```

MS_ONTM

● MS_ONTM　　IOCS コール番号：$79



機　　能　マウスボタンを押すまでの時間のチェック

入　　力

D0. L	$79	IOCS コール番号
D1. W	ボタンの指定	
D2. W	待ち時間	

リターン

D0. W　待ち時間(1～65534)
　D0. W　=0 ……ドラッグ
　D0. W　=−1 …待ち時間の最大値を超えた

解　　説

マウスボタンを押すまでの時間を調べます。
入力の D1. W は、ボタンの指定です。
0 ならば左ボタン、−1 ならば右ボタンです。
入力の D2. W は、待ち時間です。
0 ならばいつまでも待ちます。
それ以外の場合は、値が待ち時間の最大値です。

プログラム例

```
********************************************
* MS_ONTM                               $79
********************************************

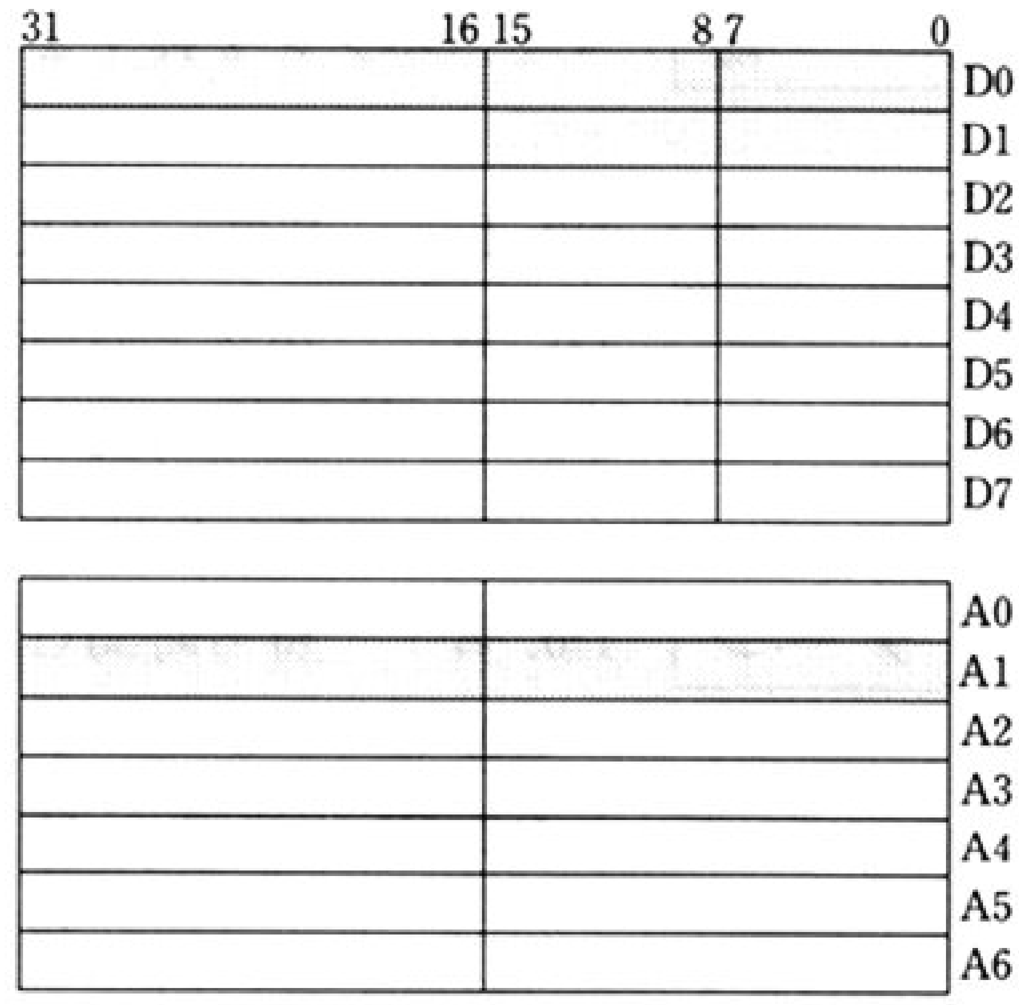
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_ONTM,d0    * IOCSコール番号をセット
        move.w  #-1,d1          * 右ボタン
        move.w  #1000,d2        *
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

MS_PATST

● MS_PATST　　IOCS コール番号：$7A



機　　能　マウスカーソルのパターン定義

入　　力

D0. L	$7A	IOCS コール番号
D1. W	カーソルの番号	
A1. L	パターンのデータ格納アドレス	
0 (A1). W	パターンの左隅からマウス座標までのドット数	
2 (A1). W	パターンの上隅からマウス座標までのドット数	
4 (A1). W	パターンの陰にするデータ（16 ワード必要）	
20 (A1). W	パターンの表示するデータ（16 ワード必要）	

リターン　なし（D0. L の値は保証されません）

解　　説　マウスカーソルのパターンを定義します。

入力の D1. W は、カーソルの番号です。

値は 0～19 です。

入力の A1. L は、パターンのデータ格納アドレスです。

入力の 0 (A1). W は、パターンの左隅からマウス座標までの距離です。

値は、0（パターンの左隅をマウス座標にする）～15（パターンの右隅をマウス座標にする）です。

入力の 2 (A1). W は、パターンの上隅からマウス座標までの距離です。

値は、0（パターンの上隅をマウス座標にする）～15（パターンの下隅をマウス座標にする）です。

入力の 4 (A1). W は、パターンの陰にするデータ（16×16 ドットのパターンなので 16 ワード必要）です。

入力の 20 (A1). W は、パターンの表示するデータ（16×16 ドットのパターンなので 16 ワード必要）です。

マウスカーソルの大きさは 16×16 ドットです。

1 ドットが 1 ビットに対応します。

プログラム例

```
*******************************************
* MS_PATST                             $7A
*******************************************

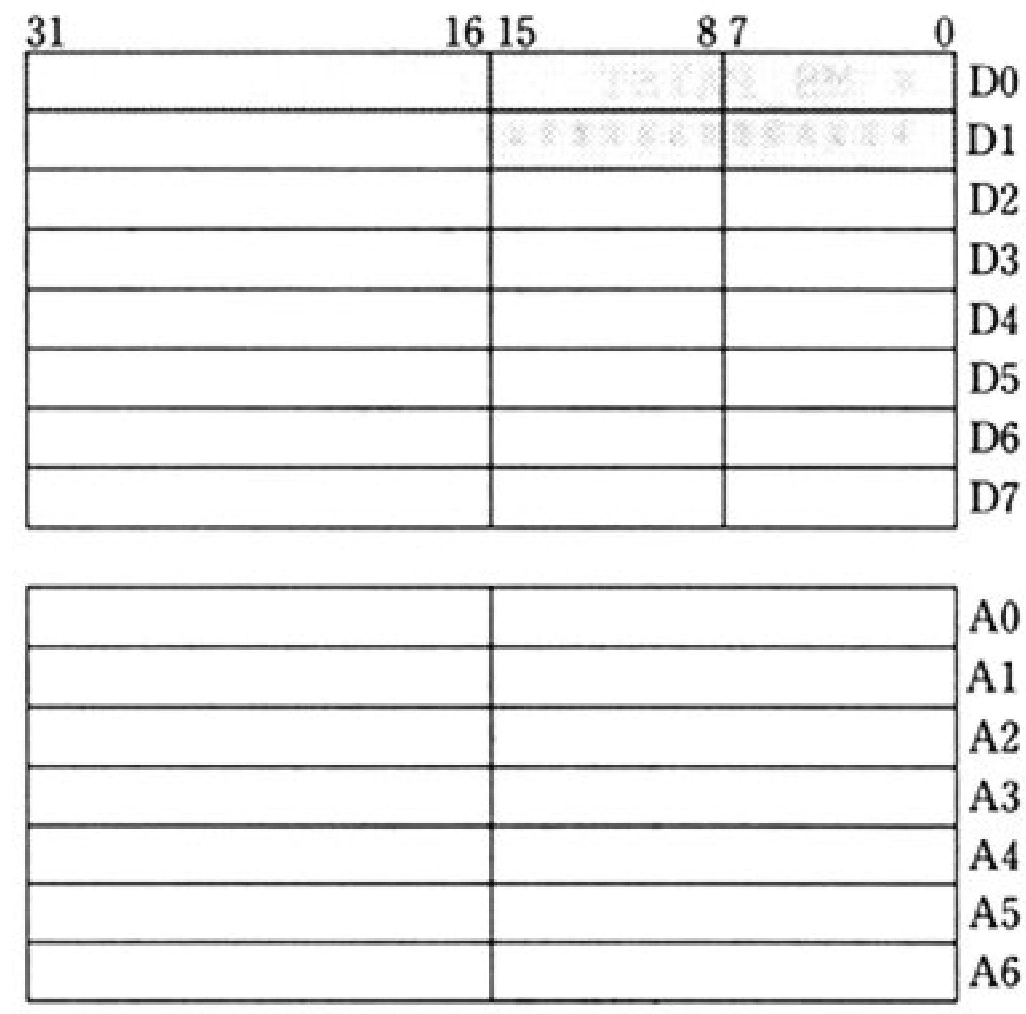
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_PATST,d0   * IOCSコール番号をセット
        move.w  #1,d1           * カーソル番号
        lea     pat1,a1         * データアドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .even
pat1:
        * マウス座標
        dc.w    0,0
        * Maskパターン 16ワード
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1111110000111111
        dc.w    %1110000000000111
        dc.w    %1111000000001111
        dc.w    %1111100000011111
        dc.w    %1111110000111111
        dc.w    %1111111001111111
        * マウスパターン 16ワード
        dc.w    %0000000000000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0000001110000000
        dc.w    %0001111111110000
        dc.w    %0000111111100000
        dc.w    %0000011111000000
        dc.w    %0000001110000000
        dc.w    %0000000100000000
        .
        .
        .end
```

MS_SEL

● MS_SEL　　IOCS コール番号：$7B



機　　能　マウスカーソルの選択

入　　力　D0. L　$7B　IOCS コール番号
D1. W　カーソルの番号
0～19 を指定します

リターン　なし（D0. L の値は保証されません）

解　　説　マウスカーソルを選択します。
入力の D1. W はカーソルの番号です。

プログラム例

```
*********************************************
* MS_SEL                                 $7B
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

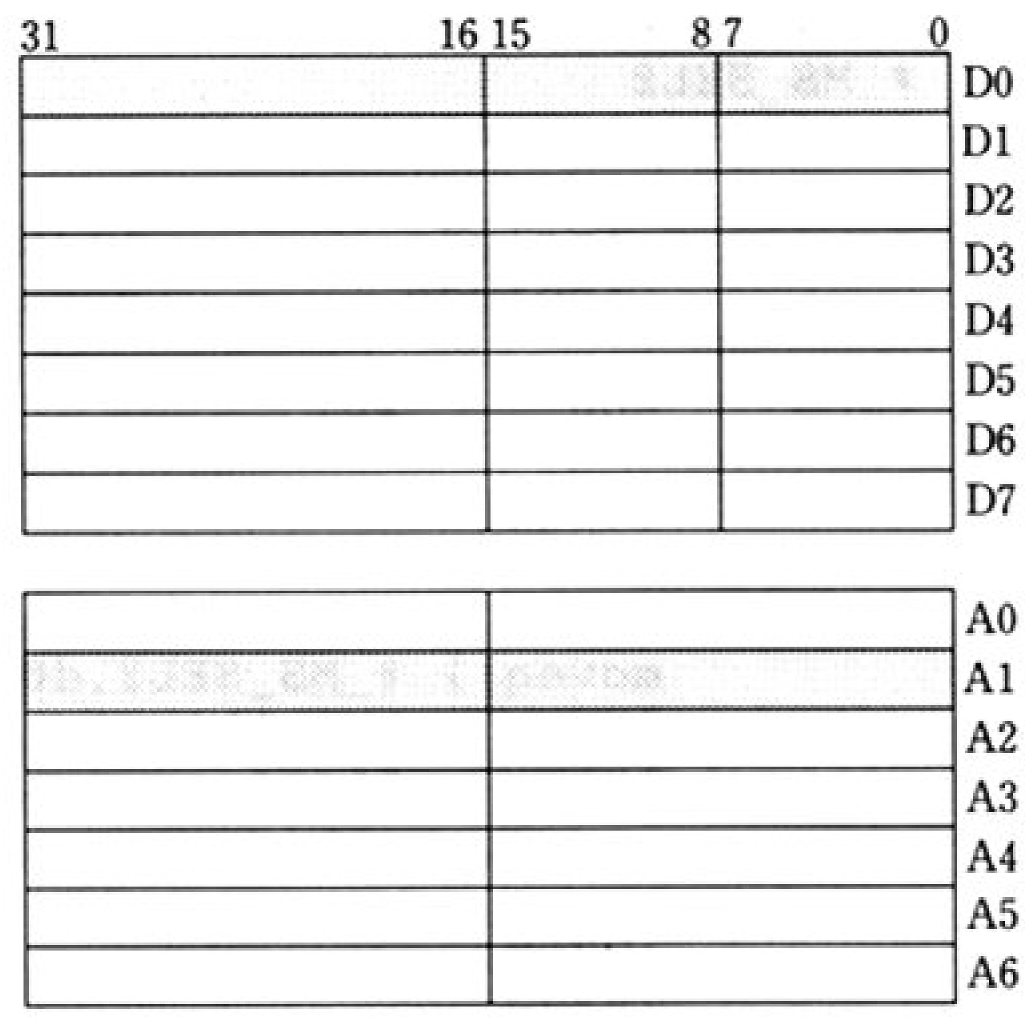
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_SEL,d0     * IOCSコール番号をセット
        move.w  #1,d1           * カーソル番号
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

MS_SEL2

● MS_SEL2　　IOCS コール番号：$7C

機　　能　マウスパターンを複数個用いたアニメーション指定



入　　力

D0. L	$7C	IOCS コール番号
A1. L	カーソルの番号テーブルアドレス	
0 (A1). W	番号 1	
2 (A1). W	番号 2	
4 (A1). W	番号 3	
⋮		
?(A1). W	−1(テーブルの終わり)	

リターン　なし (D0. L の値は保証されません)

解　　説　マウスパターンを複数個用いて、アニメーションを指定します。

入力の A1. L は、カーソルの番号テーブルアドレスです。

入力の 0 (A1). W、2 (A1). W、4 (A1). W…はそれぞれ番号 1、番号 2、番号 3…です。

入力の? (A1). W は、必ず−1でテーブルの終わりを示します。

パターンの最大数は 6 個です。

プログラム例

```
*******************************************
* MS_SEL2                               $7C
*******************************************

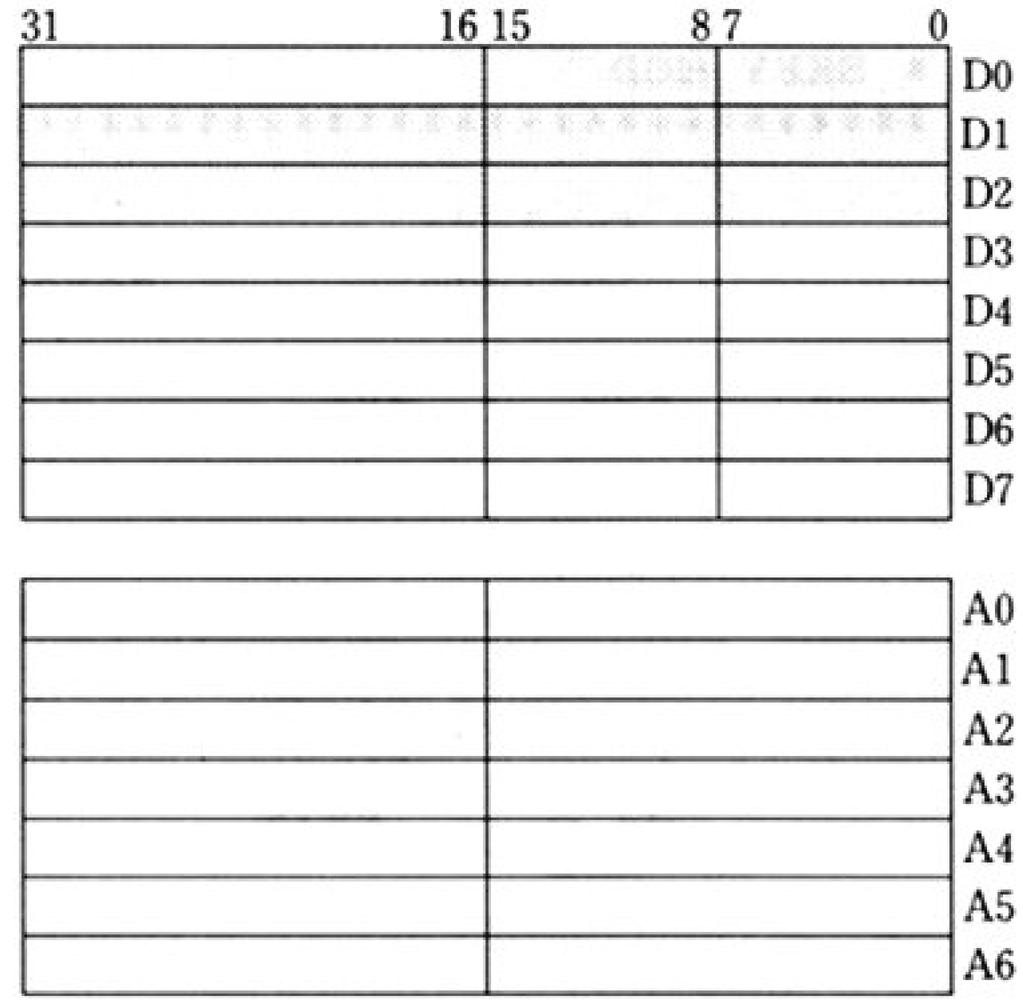
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_MS_SEL2,d0    * IOCSコール番号をセット
        lea     cur_tbl,a1      * カーソル番号テーブルアドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .even
cur_tbl:
        dc.w    1,2,3,-1        * マウスカーソル 1、2、3 を使用して
                                * アニメーションにします
        .
        .
        .end
```

SKEY_MOD

● SKEY_MOD　　IOCS コール番号：$7D



機能　ソフトキーボード制御

入力　D0. L　$7D　IOCS コール番号
D1. L　制御コード
　0……ソフトキーボード消去
　1……ソフトキーボード表示
　2……ソフトキーボード表示状態のチェック
　−1……ソフトキーボードの自動制御
D2. L　表示座標

D2. L

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
X 座標															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Y 座標															

リターン　D0. L = 1……ソフトキーボード表示状態
　= 0……ソフトキーボード消去状態

解説　ソフトキーボードを制御します。
D1. L に−1（ソフトキーボード自動制御）を指定すると、マウスの右ボタンを押すことにより状態が次のように変化します。

非表示→マウス→キーボードとマウス→マウス→非表示

入力の D2. L は D1. L が 1 のときのみ意味を持ち、表示座標をとります。
その内容は上図の通りです。
左上すみを指定します。
リターンの D0. L が 0 の場合、ソフトキーボードが消去状態であることを示します。
もし、1 の場合は表示状態であることを示します。

プログラム例 ソフトキーボードを表示します。

```
*******************************************
* SKEY_MOD                             $7D
*******************************************

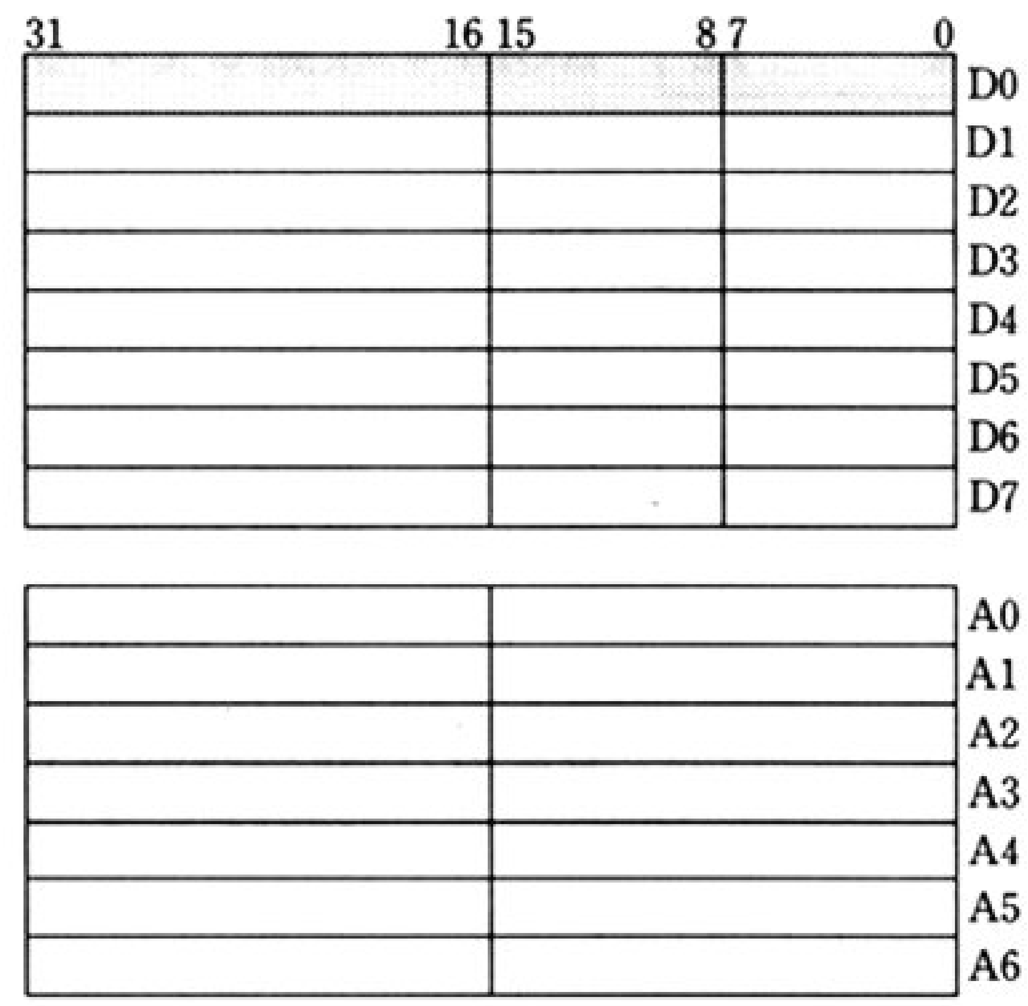
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f


        .text
        moveq.l #_SKEY_MOD,d0   * IOCSコール番号をセット
        moveq.l #1,d1           * 表示指定
        moveq.l #0,d2           * x = 0、y = 0を指定
        trap    #iocs           * IOCSコール
        dc.w    _EXIT           * プロセスの終了
        .end
```

DENSNS

● DENSNS　　IOCS コール番号：$7E



機　能　電卓のセンス

入　力　D0. L　$7E　IOCS コール番号

リターン　なし（D0. L の値は保証されません）

解　説　電卓のセンス。
答はキーボードから押されたキーとして返ります。

プログラム例　OS の電卓機能を呼び出します。

```
********************************************
* DENSNS                                 $7E
********************************************

        .include        iocscall.mac
        .include        doscall.mac

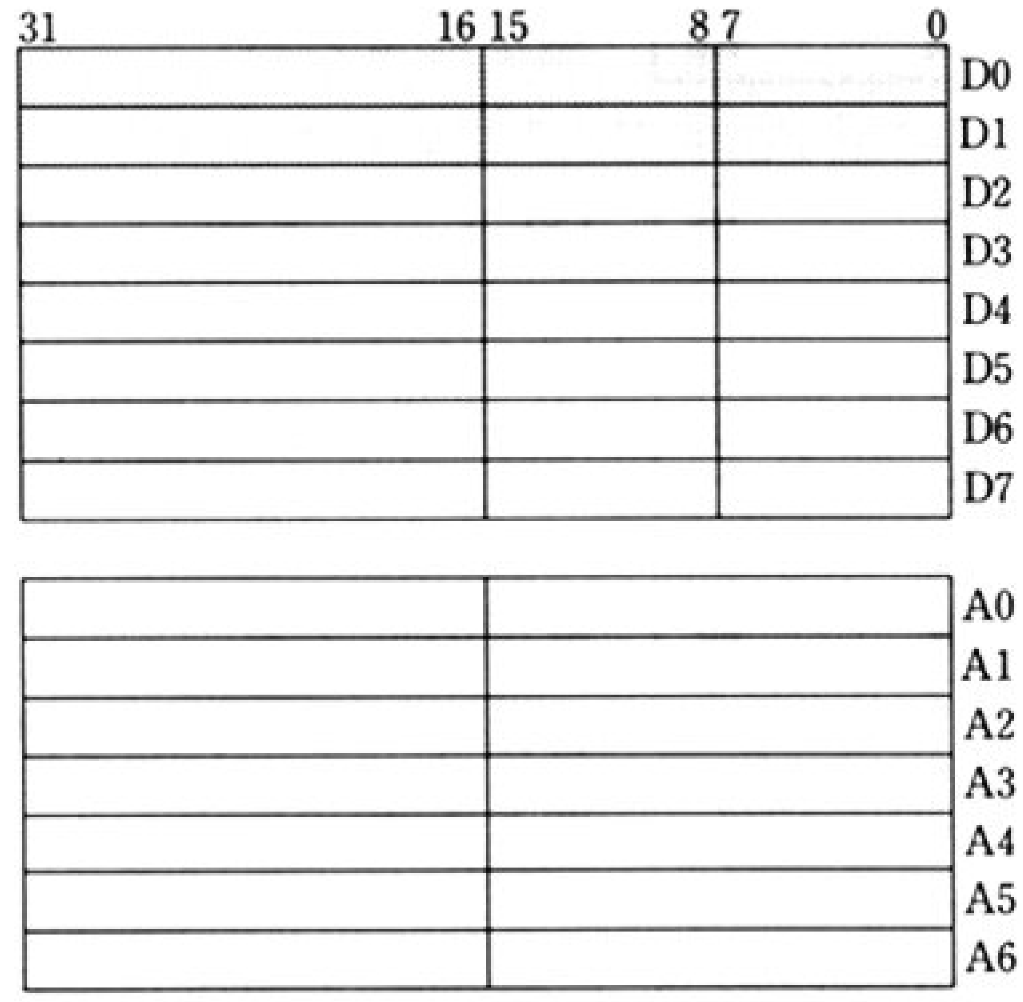
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DENSNS,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

ONTIME

● ONTIME　　IOCS コール番号：$7F

機　能　ROM が起動してからの時間を得る



入　力　D0. L　$7F　IOCS コール番号

リターン　D0. L　経過時間の時分秒(0 秒から 23 時間 59 分 59 秒 99 までの値：0～8639999(1/100 秒単位))

D1. L　経過時間の日数（0 日から 65535 日までの値：0～65535)

解　説　ROM が起動してからの時間を返します。

プログラム例

```
*****************************************
* ONTIME                            $7F
*****************************************

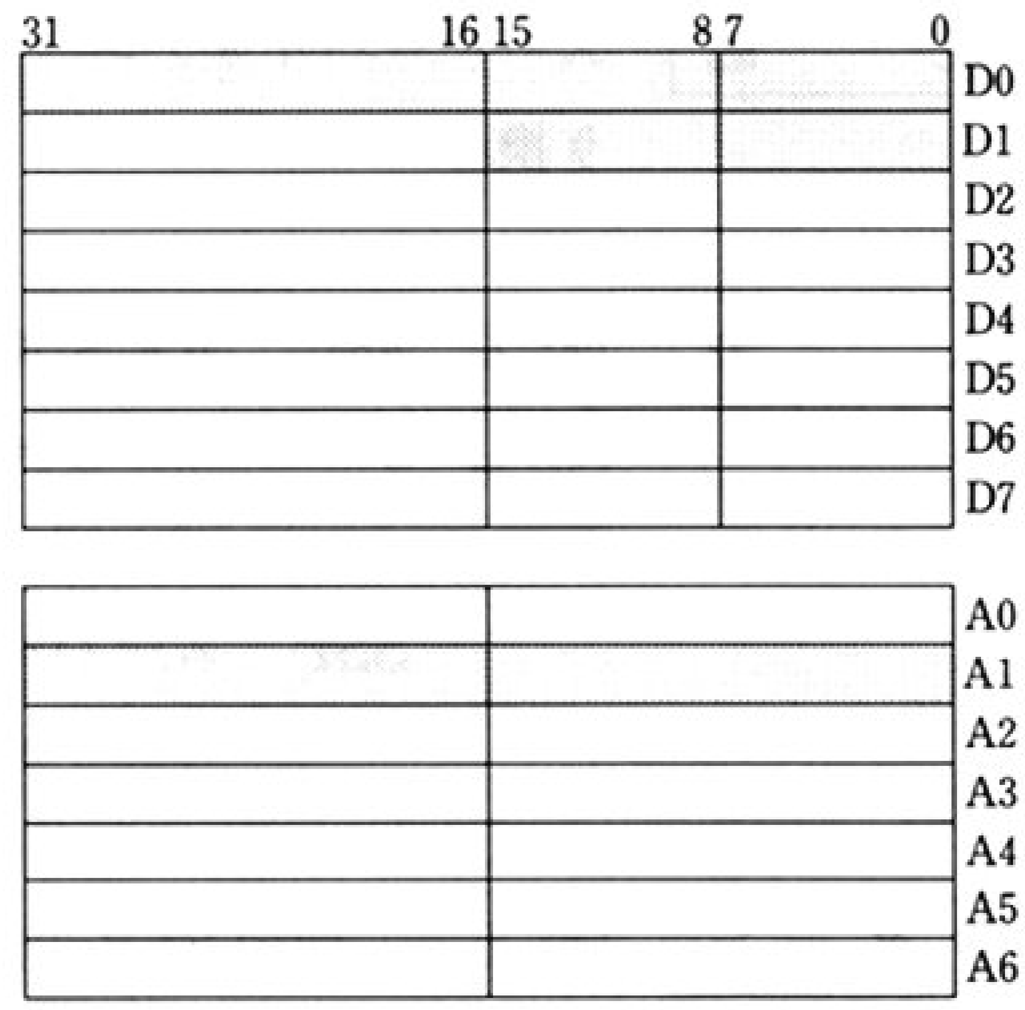
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ONTIME,d0       * IOCSコール番号をセット
        trap    #iocs             * IOCSコール
        .                         * d0.l に時間、d1.l に日数が返ってきます
        .
        .end
```

B_INTVCS

● B_INTVCS　　　IOCS コール番号：$80



機　能　ベクタの設定

入　力

D0. L	$80	IOCS コール番号
D1. W		ベクタの番号
A1. L		処理アドレス

リターン　D0. L　　設定前の処理アドレス

解　説　ベクタの設定

入力の D1. W は、ベクタの番号です。

内容を次に示します。

$0000～$00FF　=　割り込み

$0100～$01FF　=　IOCS コール

入力の A1. L は処理アドレスです。

プログラム例

```
*******************************************
* B_INTVCS                              $80
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

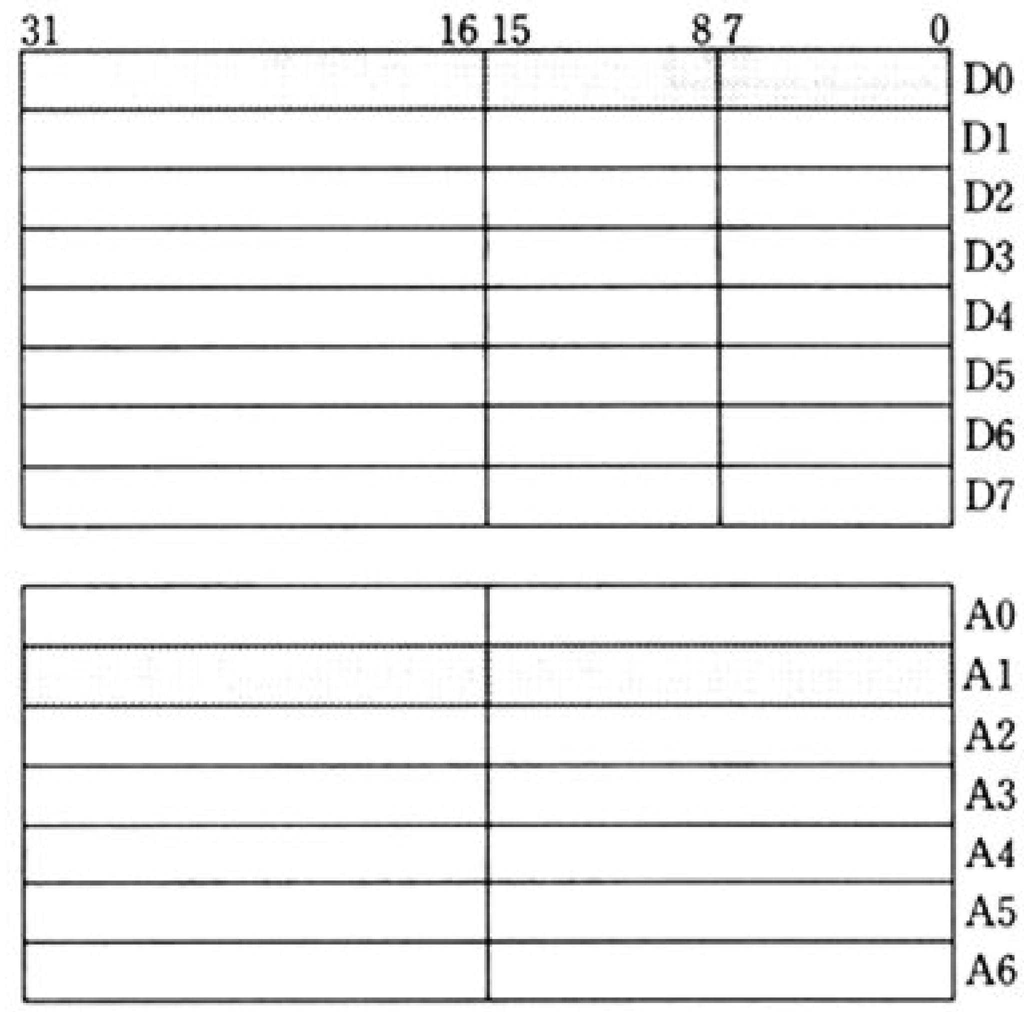
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_INTVCS,d0     * IOCSコール番号をセット
        move.w  #$01e0,d1         * ベクターの番号
        lea     user,a1           * ユーザが用意した処理ルーチンのアドレス
        trap    #iocs             * IOCSコール
        .                         * d1.l に前の処理アドレスが返ってくる
        .

user:
        movea   d0-d7/a0-a6,-(sp)         * レジスタの待避
        .                                 *
        .                                 **** 実際の処理 ****
        .                                 *
        movea   (sp)+,d0-d7/a0-a6         * レジスタの復帰
        rte                               *
        .end
*-----------------------------------------
** d1.w にセットする割り込みベクター番号は次のようになっています。
** $0000～$00ff のとき、MC68000 の例外処理ベクタ番号
** $0100～$01ff のとき、下位バイトが IOCSコール番号
```

B_SUPER

● B_SUPER　　　IOCS コール番号：$81



機能　スーパーバイザ／ユーザーモードの切り換え

入力

D0. L　$81　IOCS コール番号

A1. L　0　ユーザーモードからスーパーバイザモードへ

　　　SSP　スーパーバイザモードからユーザーモードへ

リターン

D0. L　前の SSP の値（ユーザーモードからスーパーバイザモードへ切り換えたことを示す切り換え情報）

D0. L　＝0……スーパーバイザモードからユーザーモードへ切り換えた

D0. L　＝1……エラー（スーパーパイザモードなのに、スーパーバイザモードにしようとした）

A1. L　不定

解説

スーパーバイザ／ユーザーモードの切り換えを行います。

ユーザーモードからスーパーバイザモードに切り換えるときは、A1. L に 0 を指定します。

リターンの D0. L は、SSP（システムスタックポインタ）の値です。

スーパーバイザモードからユーザーモードに戻るときは、A1. L に SSP の値を指定します。

プログラム例

```
*********************************************
* B_SUPER                               $81
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

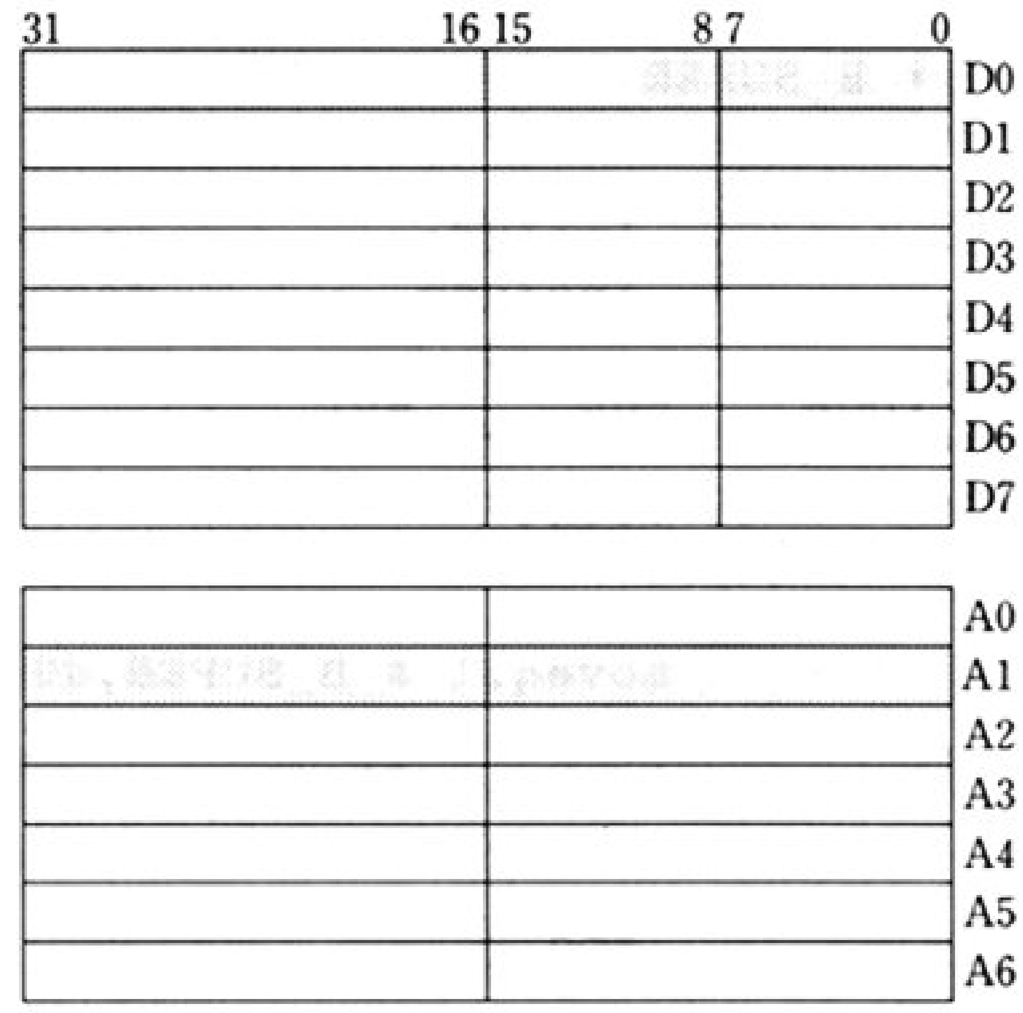
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_B_SUPER,d0     * IOCSコール番号をセット
        movea.l #0,a1            * a1.l <-- 0
        trap    #iocs            * IOCSコール
        .                        * ユーザーからスーパーに切り替わります
        .
        .end
```

B_BPEEK

● B_BPEEK　　IOCS コール番号：$82

機　能	1バイトデータ読み込み
入　力	D0. L　$82　IOCS コール番号 A1. L　読み込みアドレス
リターン	D0. B　データ A1. L　次のアドレス
解　説	指定アドレスから、1バイトデータを読み込みます。 入力の A1. L は、読み込みアドレスです。



プログラム例

```
********************************************
* B_BPEEK                               $82
********************************************

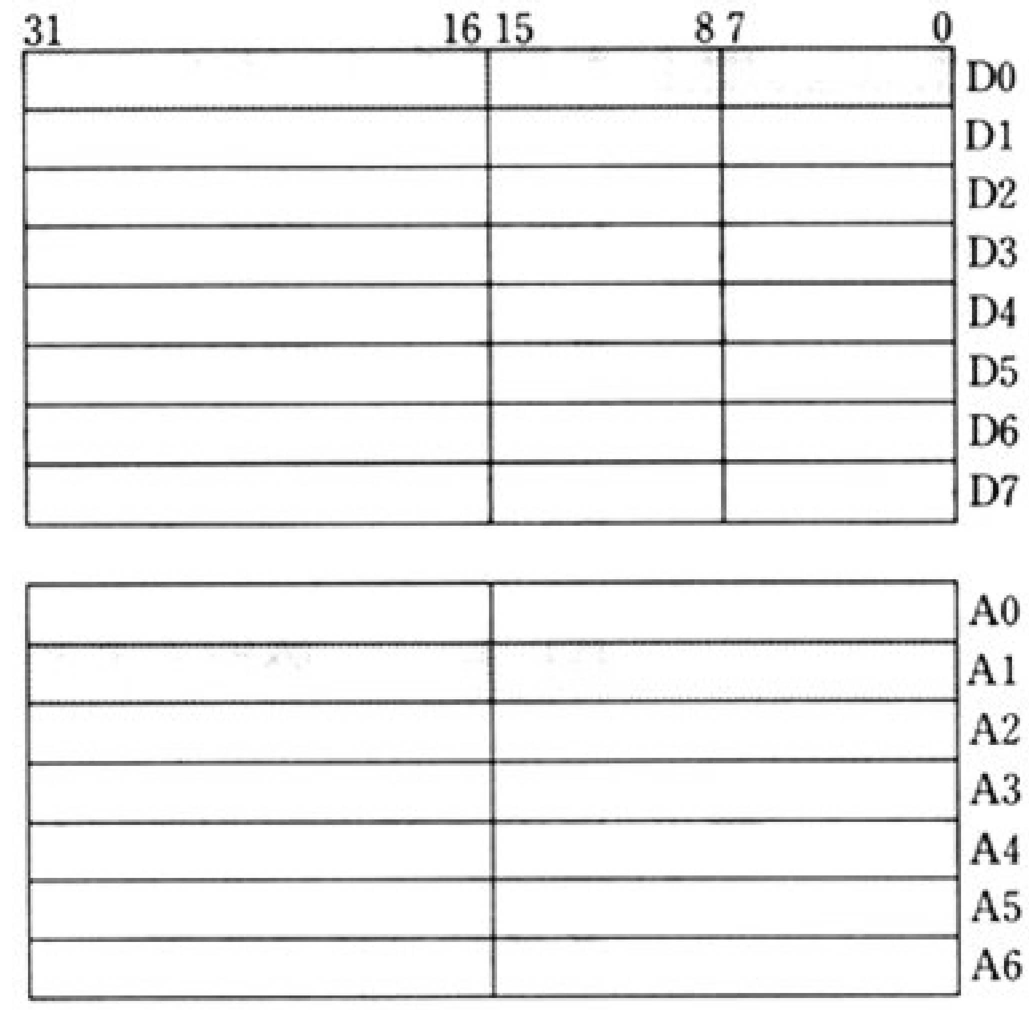
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_BPEEK,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        trap    #iocs           * IOCSコール
        .                       * d0.b データ
        .
        .end
```

B_WPEEK

● B_WPEEK　　IOCS コール番号：$83



機　　能　1ワードデータ読み込み

入　　力　D0. L　$83　IOCS コール番号
A1. L　読み込みアドレス

リターン　D0. W　データ
A1. L　次のアドレス

解　　説　指定アドレスから、1ワードデータを読み込みます。
入力の A1. L は、読み込みアドレスです。
アドレスは偶数でなければなりません。

プログラム例

```
********************************************
* B_WPEEK                              $83
********************************************

        .include        iocscall.mac
        .include        doscall.mac

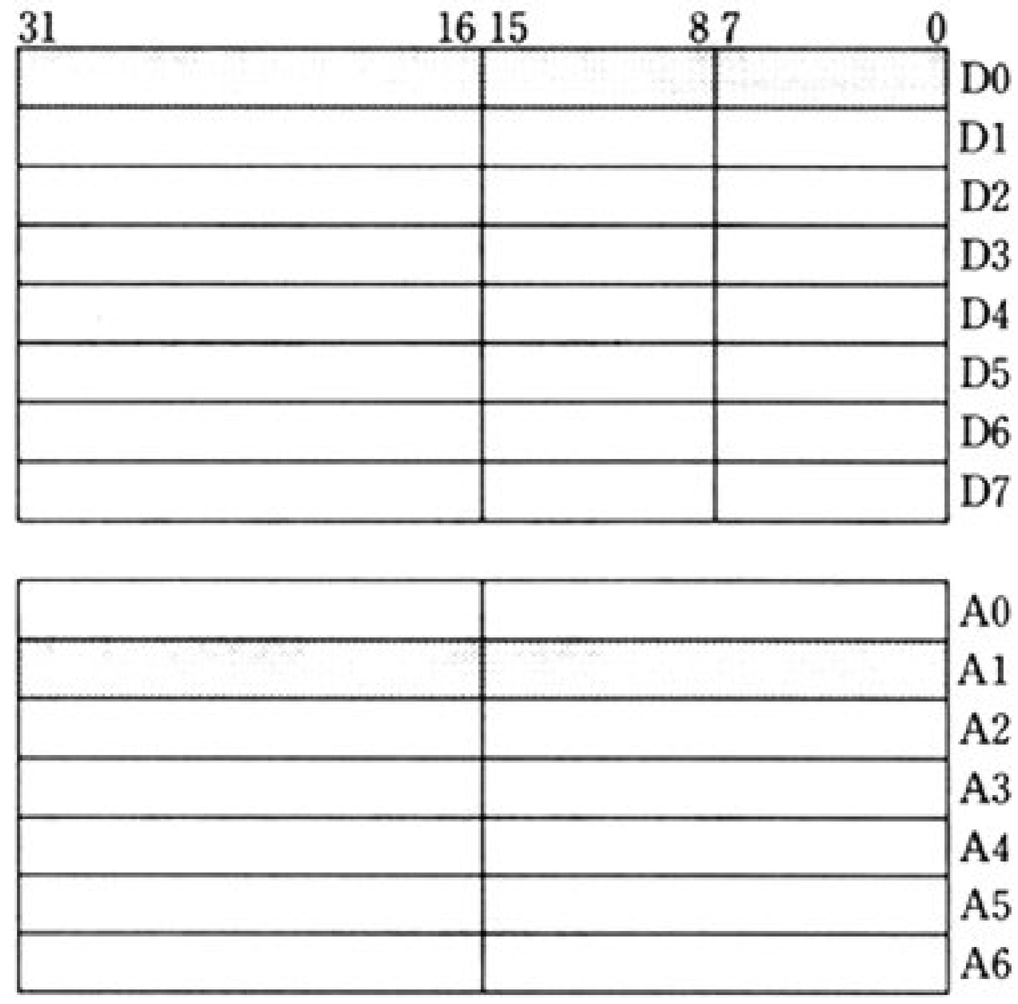
iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_WPEEK,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        trap    #iocs           * IOCSコール
        .                       * d0.w データ
        .
        .end
```

B_LPEEK

● B_LPEEK　　　IOCS コール番号：$84

機　　能	1ロングワードデータ読み込み		
入　　力	D0. L	$84	IOCS コール番号
	A1. L	読み込みアドレス	
リターン	D0. L	データ	
	A1. L	次のアドレス	



解　　説　指定アドレスから、1ロングワードデータを読み込みます。
入力の A1. L は、読み込みアドレスです。
アドレスは偶数でなければなりません。

プログラム例

```
*********************************************
* B_LPEEK                                $84
*********************************************

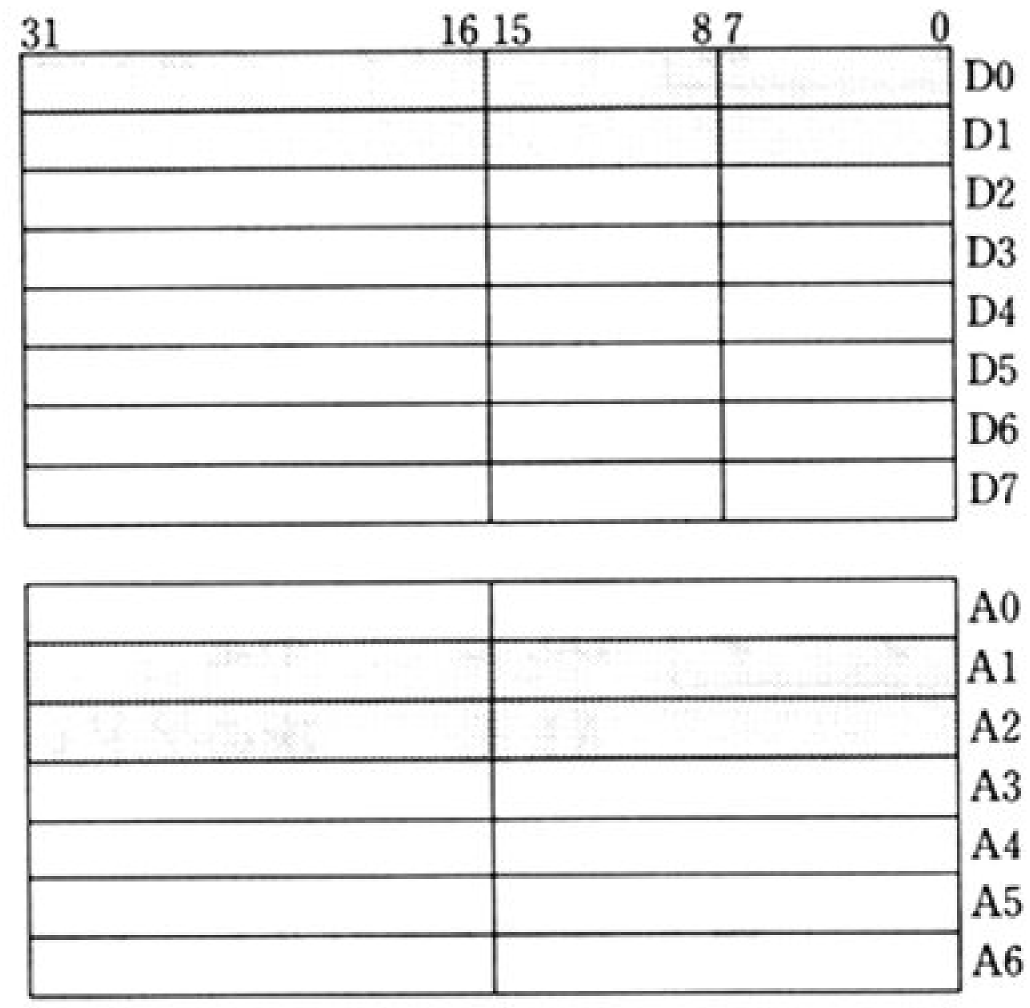
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_LPEEK,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        trap    #iocs           * IOCSコール
        .                       * d0.l データ
        .
        .end
```

B_MEMSTR

● B_MEMSTR　　　IOCS コール番号：$85



機　能　複数バイトデータ読み込み

入　力

D0. L　$85　IOCS コール番号
A1. L　読み込みアドレス（転送元）
A2. L　読み込んだデータが置かれるバッファ（転送先）
D1. L　(読み込むデータのバイト数) −1

リターン

A1. L　次のアドレス
A2. L　次のバッファアドレス

解　説　指定アドレスから、複数バイトデータを読み込みます。
バイト転送を行います。
入力の A1. L は、読み込みアドレスです。
入力の A2. L は、読み込んだデータが置かれるバッファです。
入力の D1. L は、(読み込むデータのバイト数) −1 です。

プログラム例

```
*******************************************
* B_MEMSTR                             $85
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

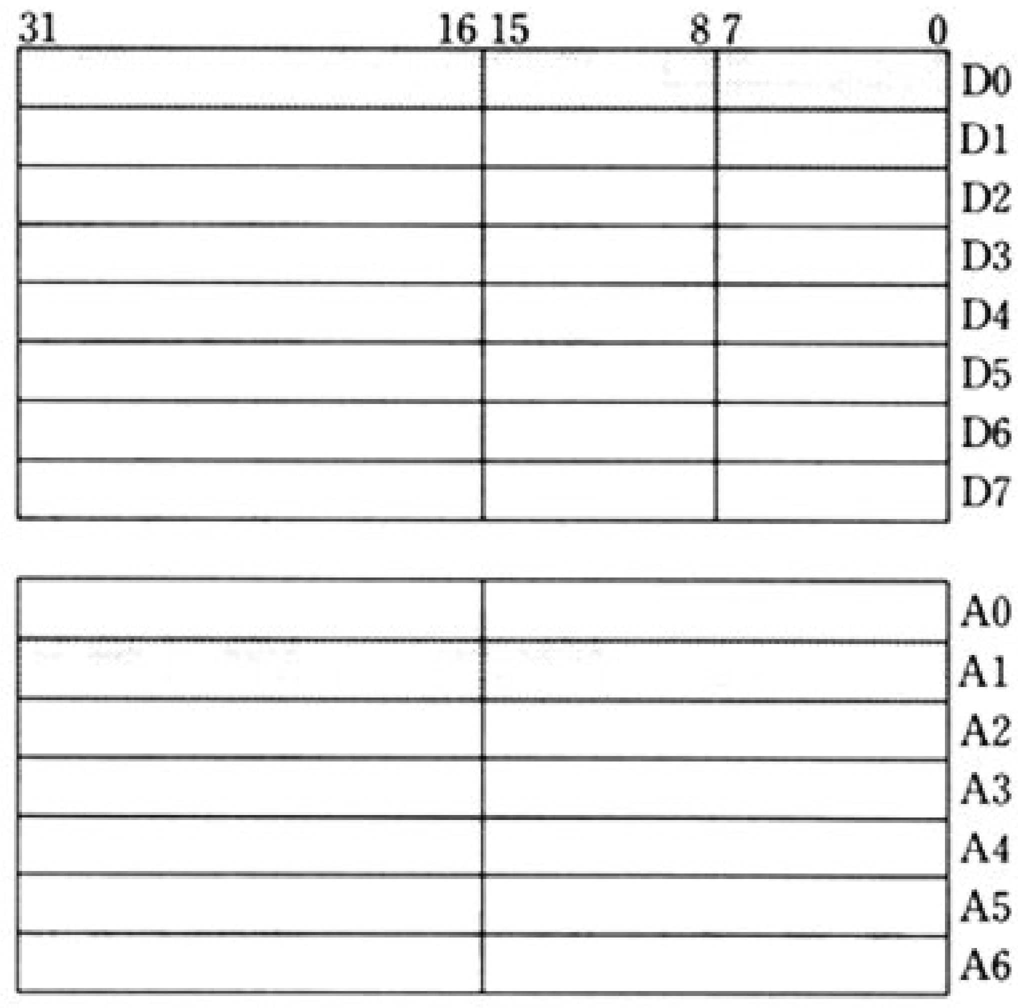
iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_MEMSTR,d0   * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        lea     buf,a2          * a2.l <-- バッファ先頭アドレス
        move.l  #$100-1,d1      * d1.l <-- バイト数-1
        trap    #iocs           * IOCSコール
        .                       * (buf) にデータが転送されます
        .

buf:    ds.b    $100
        .
        .
        .end
```

B_BPOKE

● B_BPOKE　　　IOCS コール番号：$86



機　　能	1バイトデータ書き込み
入　　力	D0. L　　$86　　IOCS コール番号 A1. L　　書き込みアドレス D1. B　　データ
リターン	D0. L　　不定 A1. L　　次のアドレス
解　　説	指定アドレスに、1バイトデータを書き込みます。 入力の A1. L は、書き込みアドレスです。 入力の D1. B はデータです。

プログラム例

```
*******************************************
* B_BPOKE                             $86
*******************************************

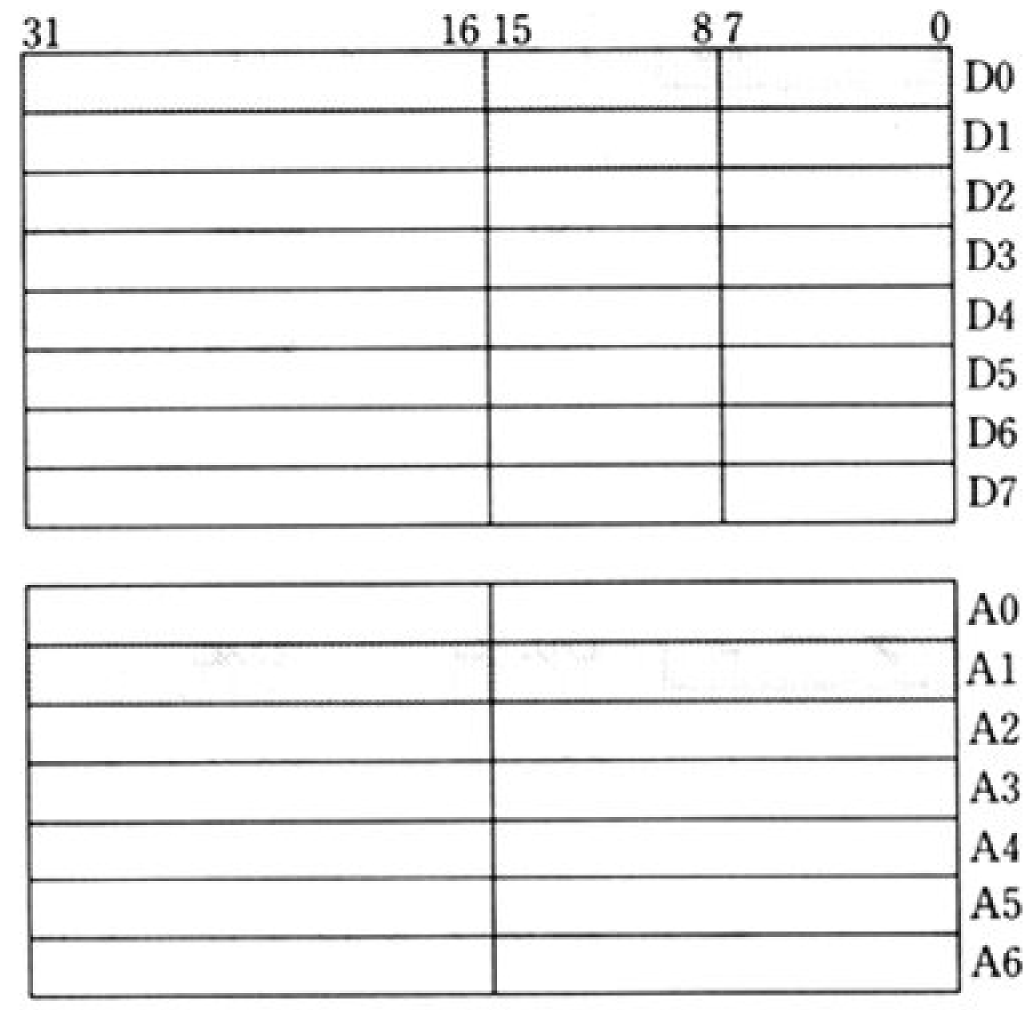
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_BPOKE,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        move.b  #0,d1           * データ 0
        trap    #iocs           * IOCSコール
        .                       * (a1).b に 0 を書き込みます
        .
        .end
```

B_WPOKE

● B_WPOKE　　IOCS コール番号：$87



機　　能　1ワードデータ書き込み

入　　力　D0. L　　$87　IOCS コール番号
A1. L　　書き込みアドレス
D1. W　　データ

リターン　D0. L　　不定
A1. L　　次のアドレス

解　　説　指定アドレスに、1ワードデータを書き込みます。
入力の A1. L は、書き込みアドレスです。
アドレスは偶数でなければなりません。
入力の D1. W はデータです。

プログラム例

```
********************************************
* B_WPOKE                               $87
********************************************

        .include        iocscall.mac
        .include        doscall.mac

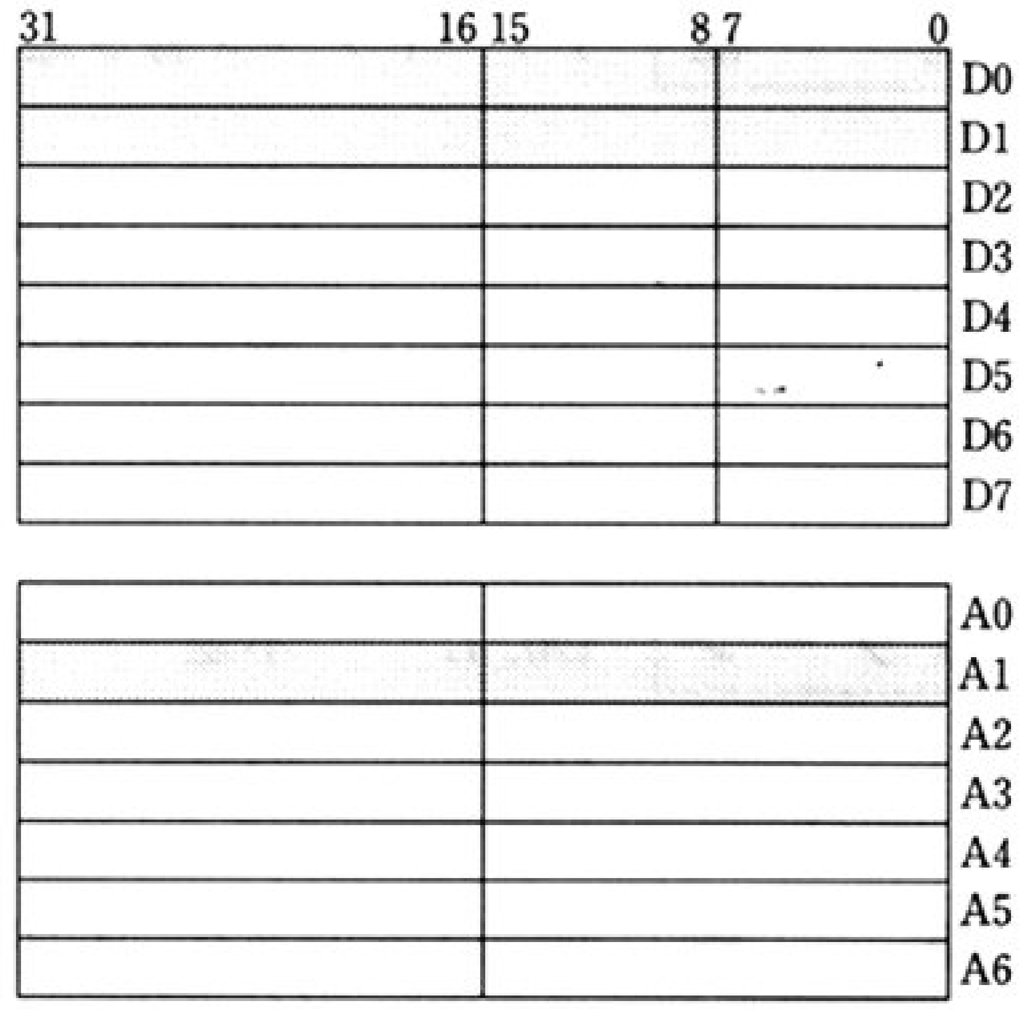
iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_WPOKE,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        move.w  #0,d1           * データ 0
        trap    #iocs           * IOCSコール
        .                       * (a1).w に 0 を書き込みます
        .
        .end
```

B_LPOKE

● B_LPOKE　　IOCS コール番号：$88

機　能	1ロングワードデータ書き込み
入　力	D0. L　$88　IOCS コール番号 A1. L　書き込みアドレス D1. L　データ
リターン	D0. L　不定 A1. L　次のアドレス
解　説	指定アドレスに、1ロングワードデータを書き込みます。 入力の A1. L は、書き込みアドレスです。 アドレスは偶数でなければなりません。 入力の D1. L はデータです。



プログラム例

```
******************************************
* B_LPOKE                            $88
******************************************

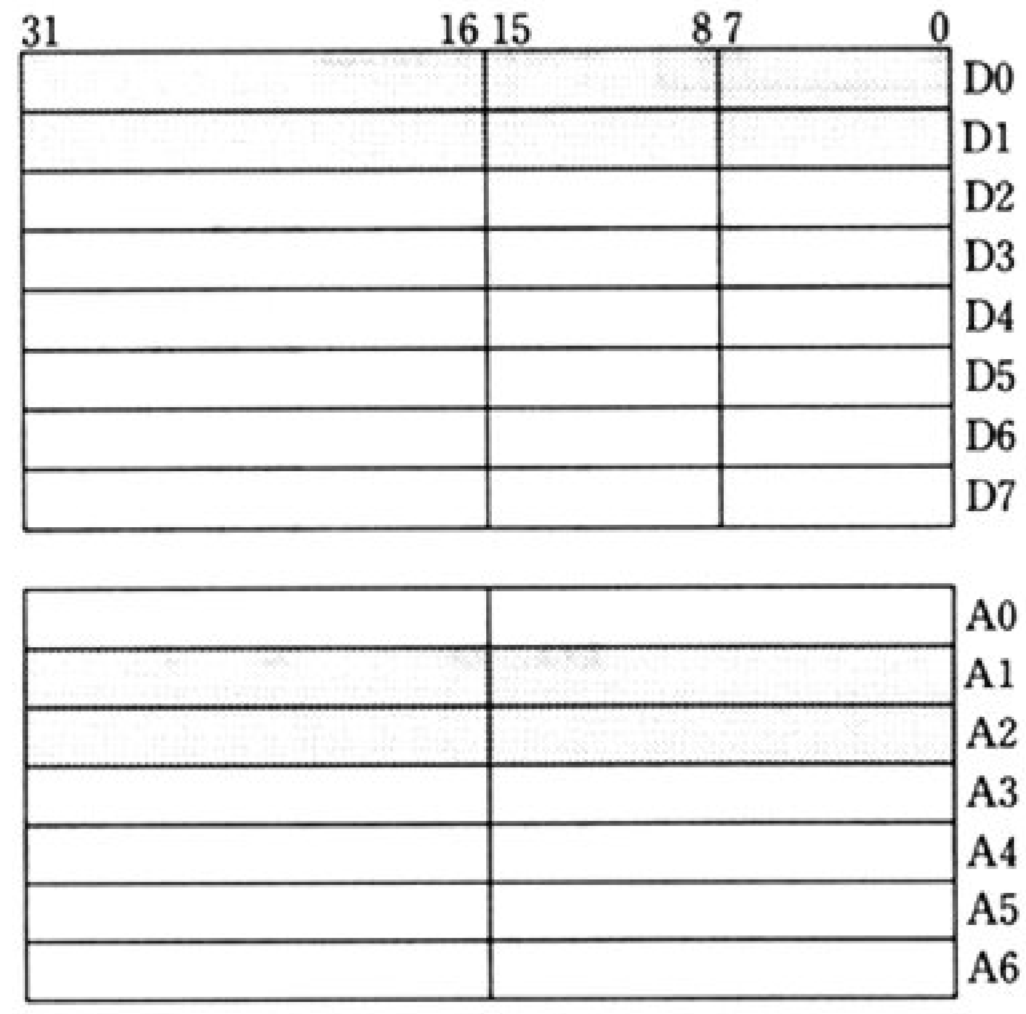
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_LPOKE,d0    * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        move.l  #0,d1           * データ 0
        trap    #iocs           * IOCSコール
        .                       * (a1).l に 0 を書き込みます
        .
        .end
```

B_MEMSET

● B_MEMSET　　IOCS コール番号：$89



機　能　複数バイトデータ書き込み

入　力

D0. L	$89	IOCS コール番号
A1. L	書き込みアドレス(転送先)	
A2. L	書き込むべきデータが置かれているバッファ(転送元)	
D1. L	(書き込むデータのバイト数)−1	

リターン

D0. L	不定
A1. L	次のアドレス
A2. L	次のバッファアドレス

解　説　指定アドレスに、複数バイトデータを書き込みます。
バイト転送を行います。
入力の A1. L は、書き込みアドレスです。
入力の A2. L は、書き込むべきデータが置かれているバッファです。
入力の D1. L は、(書き込むデータのバイト数)−1 です。

プログラム例

```
*******************************************
* B_MEMSET                            $89
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

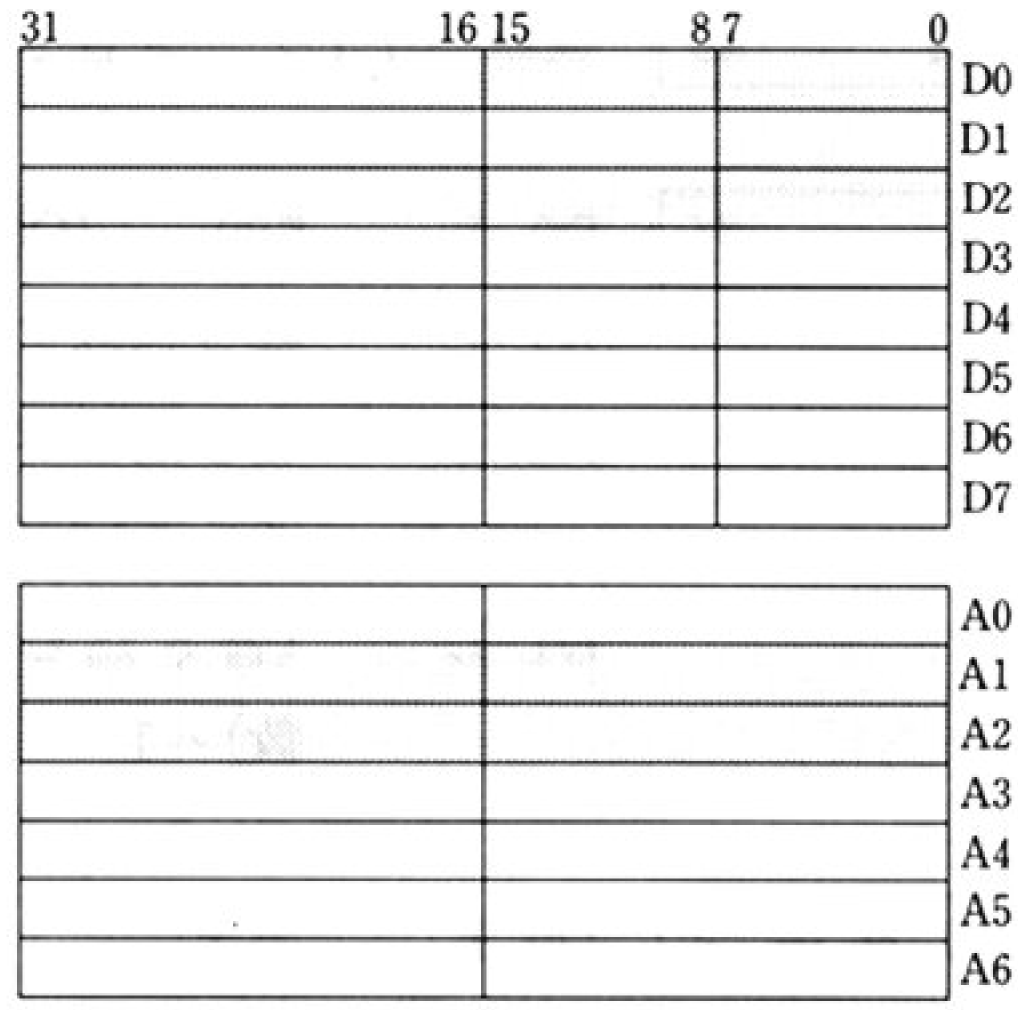
iocs    equ     $0f
SUPER   equ     $e00000         * テキストVRAM

        .text
        .
        .
        moveq.l #_B_MEMSET,d0   * IOCSコール番号をセット
        lea     SUPER,a1        * スーパーバイザー領域のアドレスのロード
        lea     buf,a2          * a2.l <-- 転送元アドレス
        move.l  #16-1,d1        * d1.l <-- バイト数-1
        trap    #iocs           * IOCSコール
        .                       * (SUPER) にデータが転送されます
        .

buf:    dc.b    "System Message ",0
        .
        .
        .end
```

DMAMOVE

● DMAMOVE　　　IOCS コール番号：$8A



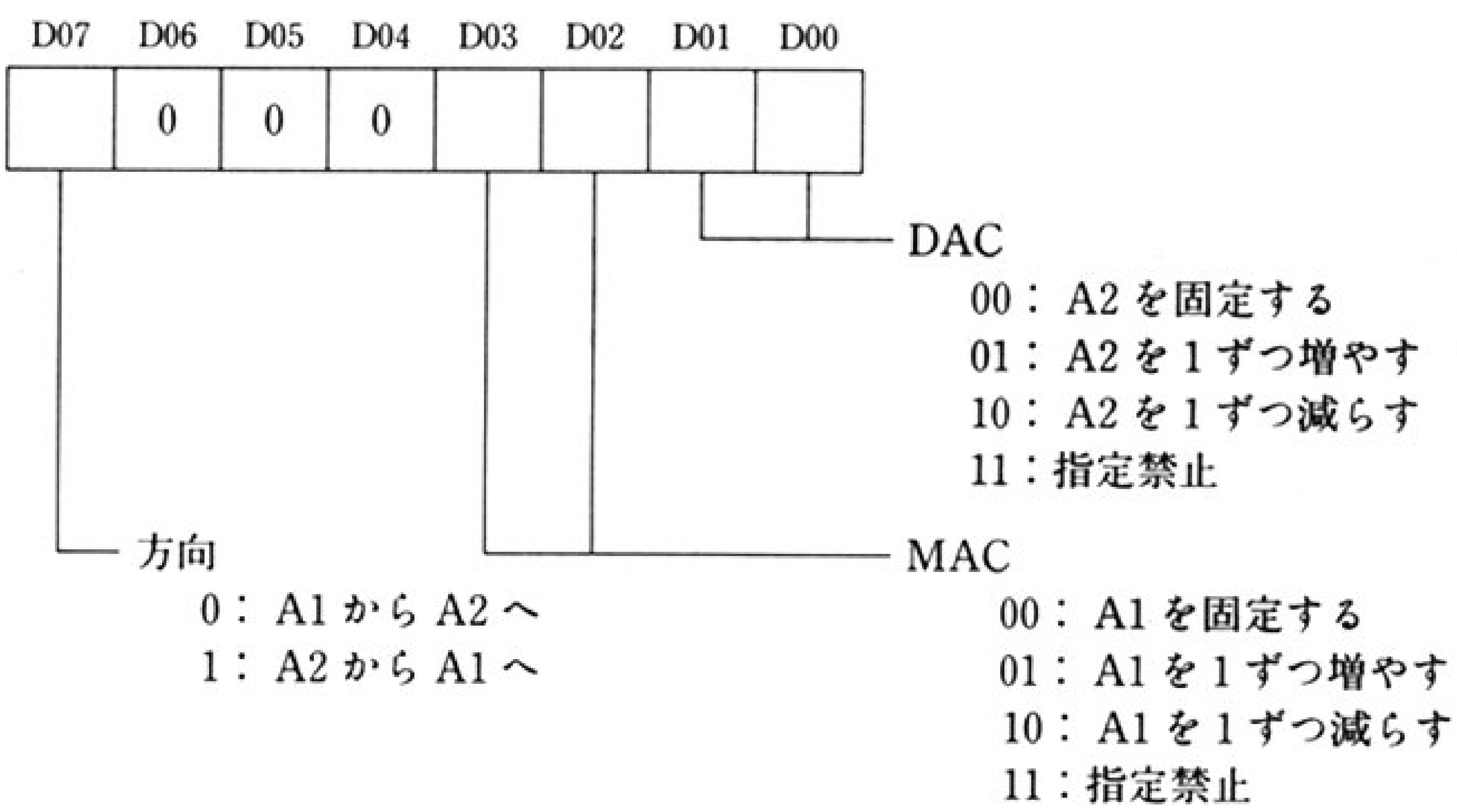
機　能　DMA 転送

入　力

D0. L	$8A	IOCS コール番号
A1. L	転送元アドレス	
D2. L	転送データのバイト数	
A2. L	転送先アドレス	
D1. B	モード	

リターン　なし（D0. L の値は保証されません）

解　説　DMA 転送を行います。
長さは$FF00 以内です。
入力の A1. L は、転送元アドレスです。
入力の D2. L は、転送データのバイト数です。
入力の A2. L は、転送先アドレスです。
入力の D1. B は、モードです。
内容は次の通りです。



プログラム例

```
*******************************************
* DMAMOVE                              $8A
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

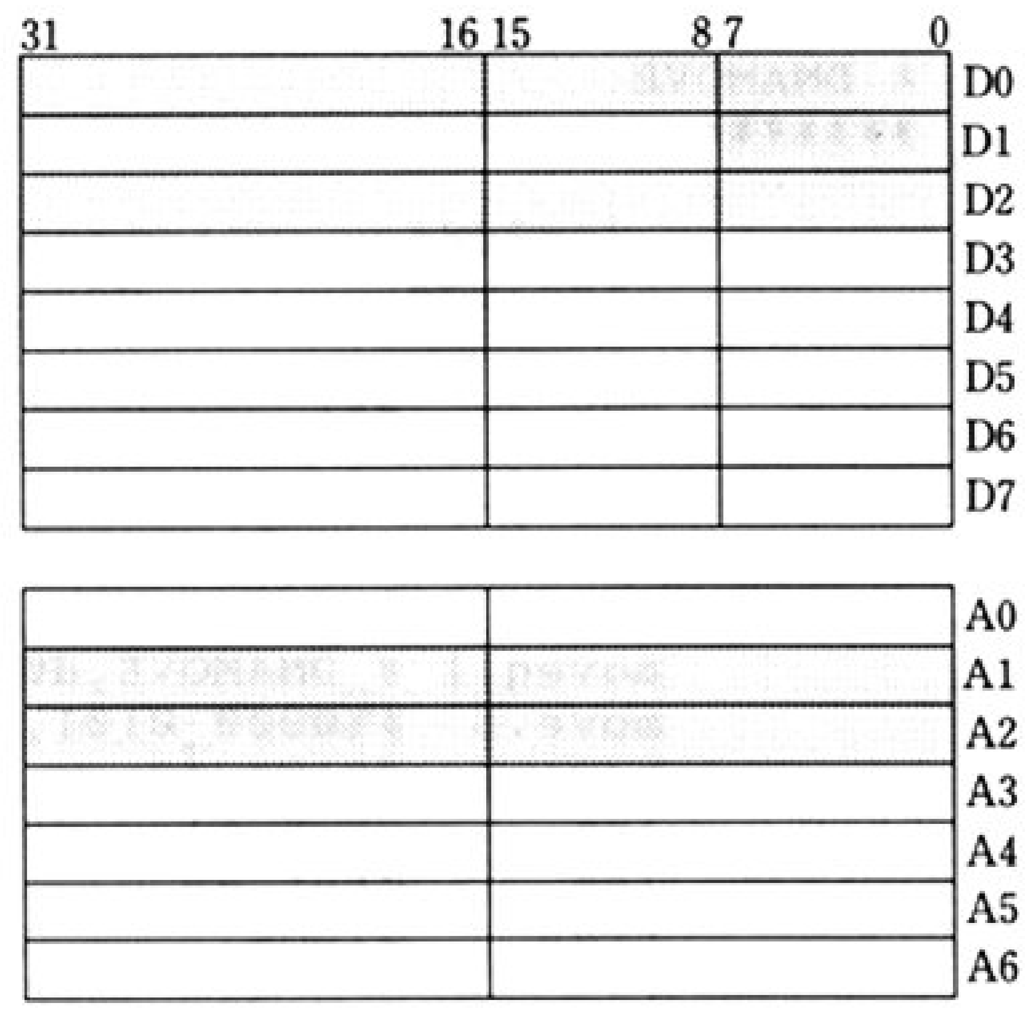
iocs    equu    $0f

        .text
        .
        .
        moveq.l #_DMAMOVE,d0    * IOCSコール番号をセット
        move.b  #%0000_0101,d1  * モード
        move.l  #$100,d2        * 転送バイト数 $100
        lea     sbuf,a1         * 転送元アドレス
        lea     dbuf,a2         * 転送先アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
sbuf:   dcb.b   $100,$20
dbuf:   ds.b    $100
        .
        .
        .end
```

DMAMOV_A

● **DMAMOV_A　　IOCS コール番号：$8B**

機　能　アレイチェーンによる DMA 転送



入　力

D0. L	$8B　IOCSコール番号
A1. L	転送元データチェーン テーブルアドレス
0 (A1). L	先頭アドレス 1
4 (A1). W	長さ 1
6 (A1). L	先頭アドレス 2
10 (A1). W	長さ 2
⋮	
?? (A1). L	先頭アドレス N
?? (A1). W	長さ N
D2. L	転送データチェーンテーブルの個数
A2. L	転送先アドレス
D1. B	モード

リターン　なし (D0. L の値は保証されません)

解　説　アレイチェーンによる DMA 転送を行います。

入力の A1. L は、転送元データチェーンテーブルアドレスです。
入力の 0 (A1). L は、先頭アドレス 1 です。
入力の 4 (A1). W は、長さ 1 です。
長さは、$0001～$FFFF の値です。
入力の 6 (A1). L は、先頭アドレス 2 です。
入力の 10 (A1). W は、長さ 2 です。
長さは、$0001～$FFFF の値です。
⋮
入力の?? (A1). L は、先頭アドレス N です。
入力の?? (A1). W は、長さ N です。
長さは、$0001～$FFFF の値です。
入力の D2. L は、転送データチェーンテーブルの個数 N です。
入力の A2. L は、転送先アドレスです。
入力の D1. B は、モードです。
内容については、DMAMOVE ($8A) を参照してください。

プログラム例

```
*******************************************
* DMAMOV_A                              $8B
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

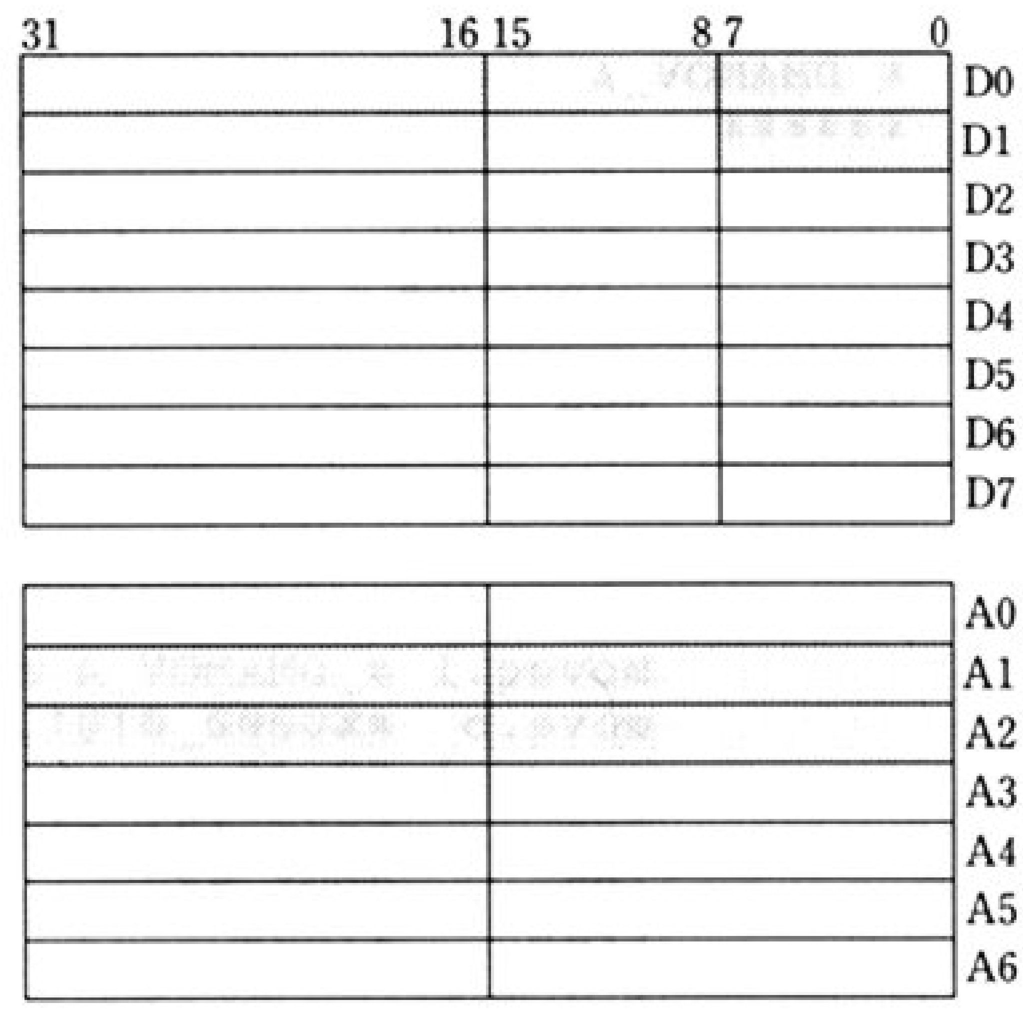
        .text
        .
        .
        moveq.l #_DMAMOV_A,d0   * IOCSコール番号をセット
        move.b  #%0000_0101,d1  * モード
        moveq.l #2,d2           * 転送元データチェーンテーブルの個数
        lea     aray_tbl,a1     * 転送元データチェーンテーブルアドレス
        lea     dbuf,a2         * 転送先アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .even
aray_tbl:
        dc.l    adr1            * 転送元アドレス 1
        dc.w    $ff             * 転送バイト数 1
        dc.l    adr2            * 転送元アドレス 2
        dc.w    $ff             * 転送バイト数 2

adr1:   dcb.b   $ff,'1'
adr2:   dcb.b   $ff,'2'
        .
        .
dbuf:   ds.b $ff+$ff
        .
        .
        .end
```

DMAMOV_L

● DMAMOV_L　　IOCS コール番号：$8C

機　能　リンクアレイチェーンによるDMA転送



入　力

D0. L	$8C　IOCS コール番号
A1. L	転送元データチェーンテーブルアドレス
0 (A1). L	先頭アドレス 1
4 (A1). W	長さ 1
6 (A1). L	次のテーブルアドレス 1
0 (?1). L	先頭アドレス 2 (?1 はテーブルアドレス 1)
4 (?1). W	長さ 2
6 (?1). L	次のテーブルアドレス 2
0 (?(N-1)). L	先頭アドレス N (?(N-1) はテーブルアドレス(N-1))
4 (?(N-1)). W	長さ N
6 (?(N-1)). L	0　テーブル終わり
A2. L	転送先アドレス
D1. B	モード

リターン　なし (D0. L の値は保証されません)

解　説　リンクアレイチェーンによるDMA転送を行います。

入力の A1. L は、転送元データチェーンテーブルアドレスです。

入力の 0 (A1). L は、先頭アドレス 1 です。

入力の 4 (A1). W は、長さ 1 です。

長さは、$0001～$FFFF の値です。

入力の 6 (A1). L は、次のテーブルアドレス 1 です。

入力の 0 (?1). L は、先頭アドレス 2 (?1 はテーブルアドレス 1)です。

入力の 4 (?1). W は、長さ 2 です。

長さは、$0001～$FFFF の値です。

入力の 6 (?1). L は、次のテーブルアドレス 2 です。

入力の 0 (?(N-1)). L は、先頭アドレス N (?(N-1)はテーブルアドレス(N-1))です。

入力の 4 (?(N-1)). W は、長さ N です。

長さは、$0001～$FFFF の値です。

入力の 6 (?(N-1)). L は、つねに 0 で、テーブル終わりを示します。

入力の A2. L は、転送先アドレスです。

入力のD1.Bは、モードです。

内容については、DMAMOVE($8A)を参照してください。

プログラム例

```
*******************************************
* DMAMOV_L                               $8C
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

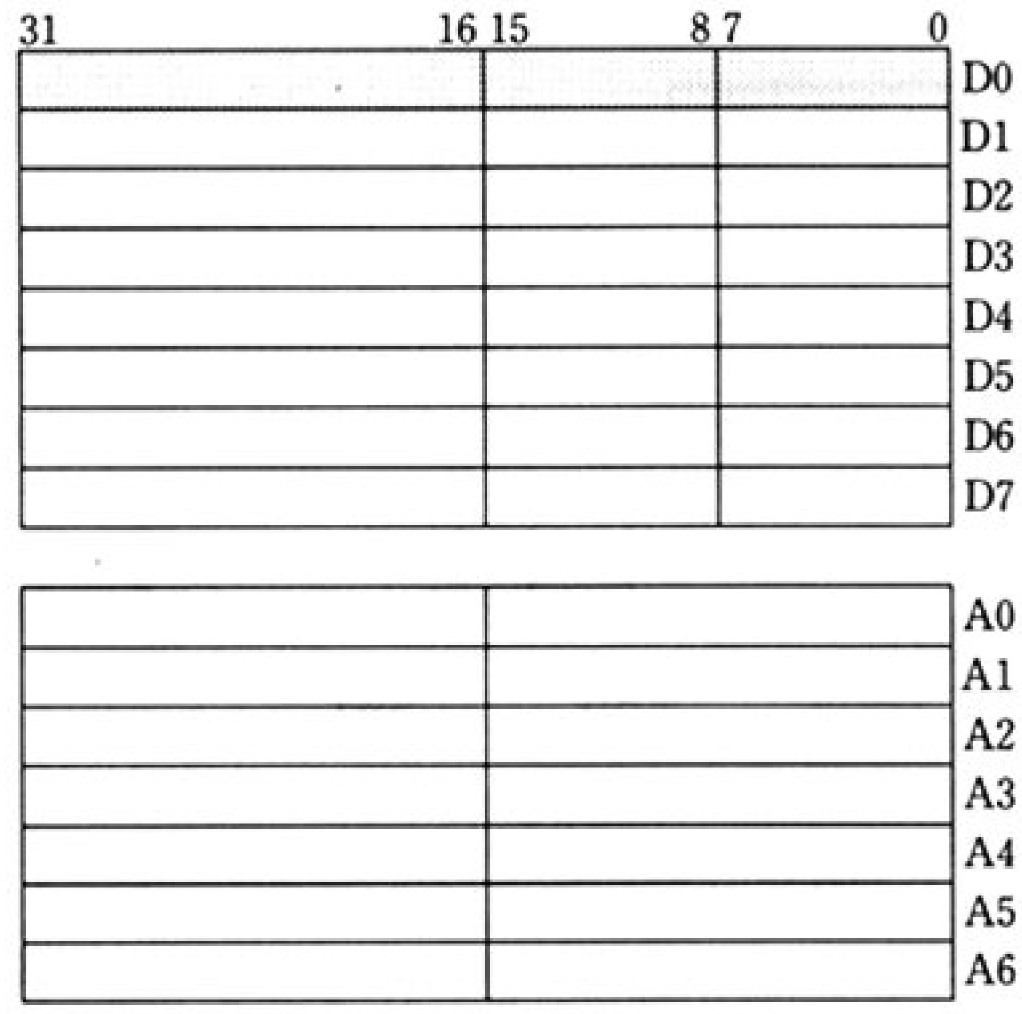
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DMAMOV_L,d0   * IOCSコール番号をセット
        move.l  #%0000_0101,d1  * モード
        moveq.l #2,d2           * 転送元データチェーンテーブルの個数
        lea     link_tbl_1,a1   * 転送元データチェーンテーブルアドレス
        lea     dbuf,a2         * 転送先アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .even
link_tbl_1:
        dc.l    adr1            * 転送元アドレス 1
        dc.w    $ff             * 転送バイト数 1
        dc.l    link_tbl_2      * 次のテーブルアドレス 1 (link_tbl_2)

link_tbl_2:
        dc.l    adr2            * 転送元アドレス 2
        dc.w    $ff             * 転送バイト数 2
        dc.l    0               * 次のテーブルアドレス 2
adr1:   dcb.b   $ff,'1'
adr2:   dcb.b   $ff,'2'
        .
        .
dbuf    ds.b    $ff+$ff
        .
        .
        .end
```

DMAMODE

● DMAMODE　　IOCS コール番号：$8D



機　　能　DMA の実行モードセンス

入　　力　D0. L　$8D　IOCS コール番号

リターン　D0. L　実行モード

D0. L　=$00…何もしていない

D0. L　=$8A…転送中(DMAMOVE($8A)を実行中)

D0. L　=$8B…転送中(DMAMOV_A($8B)を実行中)

D0. L　=$8C…転送中(DMAMOV_L($8C)を実行中)

解　　説　DMA の実行モードをセンスします。

プログラム例

```
********************************************
* DMAMODE                               $8D
********************************************

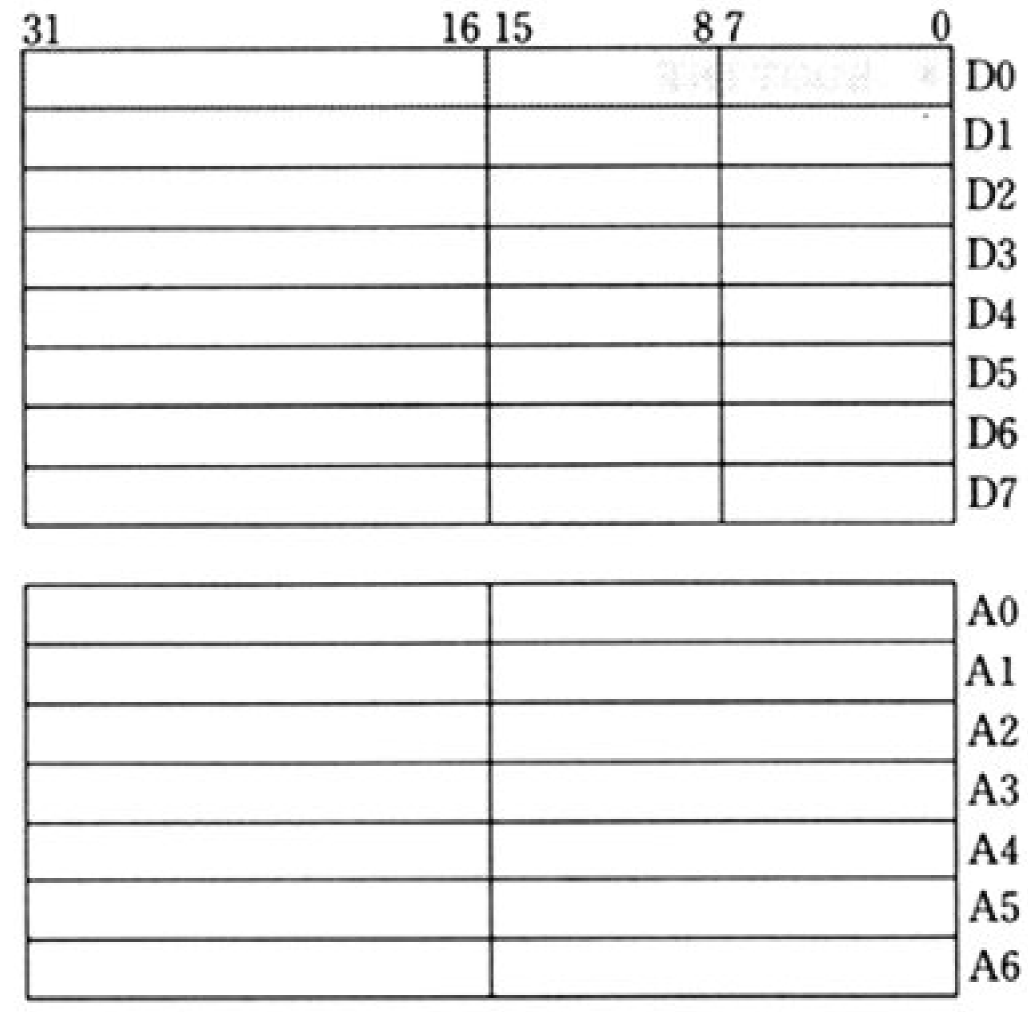
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_DMAMODE,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

BOOTINF

● BOOTINF　　　IOCS コール番号：$8E



機　　能　ブート情報を得る

入　　力　D0. L　　$8E　IOCSコール番号

リターン　D0. L　　ブート情報

解　　説　ブート情報を返します。

リターンのD0. Lは、ブート情報です。

内容は次のとおりです。

ビット31～24：起動情報

0……パワースイッチにより起動

1……外部スイッチにより起動

2……タイマにより起動

ビット23～0：ブート情報

$80～$8F ………………ハードディスクからブート（下位4ビットがブートしたドライブの番号を表す）

$90～$93 ………………2HDディスクからブート（下位2ビットがブートしたドライブの番号を表す）

$ED0000～$DE3FFE … SRAMからブート（値はブートアドレス）

上記以外 ………………ROMからブート

プログラム例

```
*******************************************
* BOOTINF                               $8E
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_BOOTINF,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .                       * d0.l にブート情報が返ってきます
        .
        .end
```

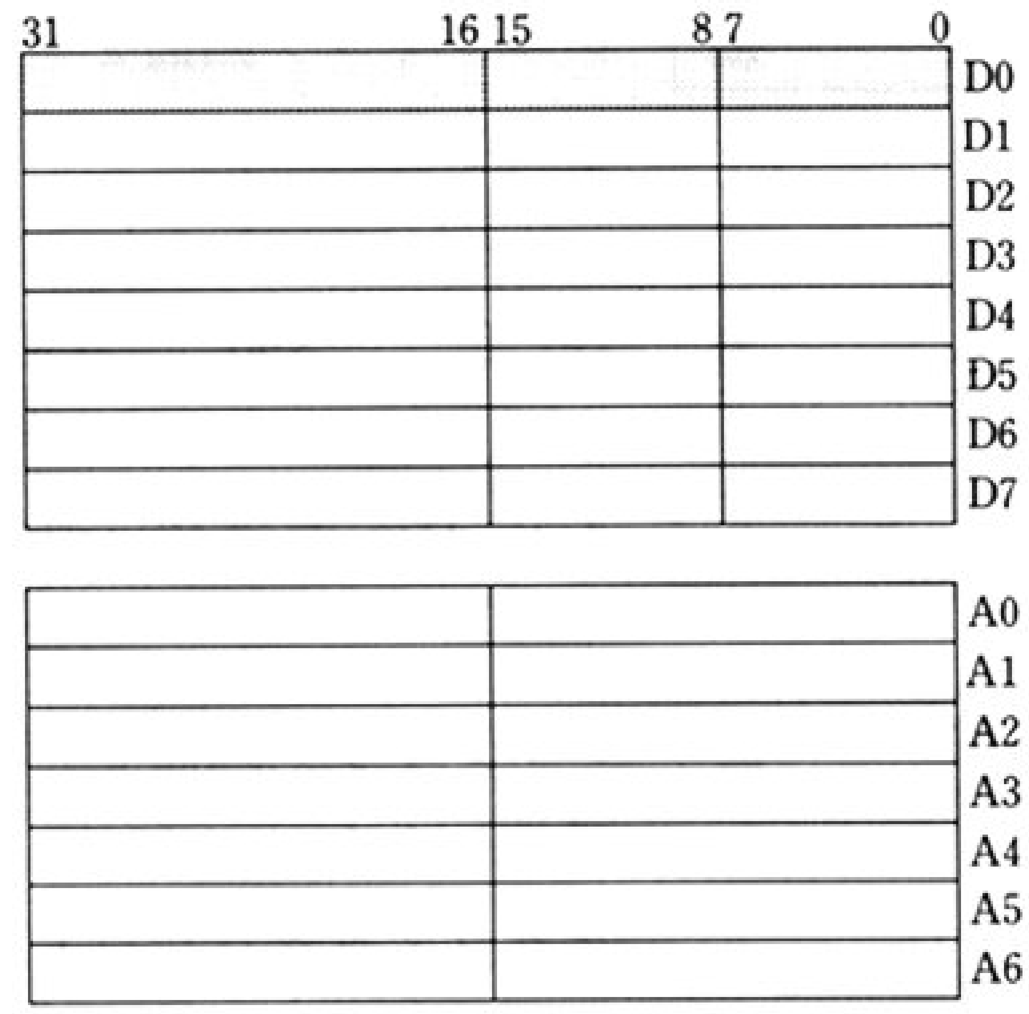
ROMVER

● ROMVER　　IOCSコール番号：$8F

機　能　ROMのバージョンと作成年月日を得る

入　力　D0.L　$8F　IOCSコール番号

リターン　D0.L　バージョンと作成日
(BCD表現)



D0.L

D31–D24	D23–D16
バージョン	年

D15–D08	D07–D00
月	日

解　説　ROMのバージョンと作成年月日を返します。

プログラム例

```
*********************************************
* ROMVER                                 $8F
*********************************************

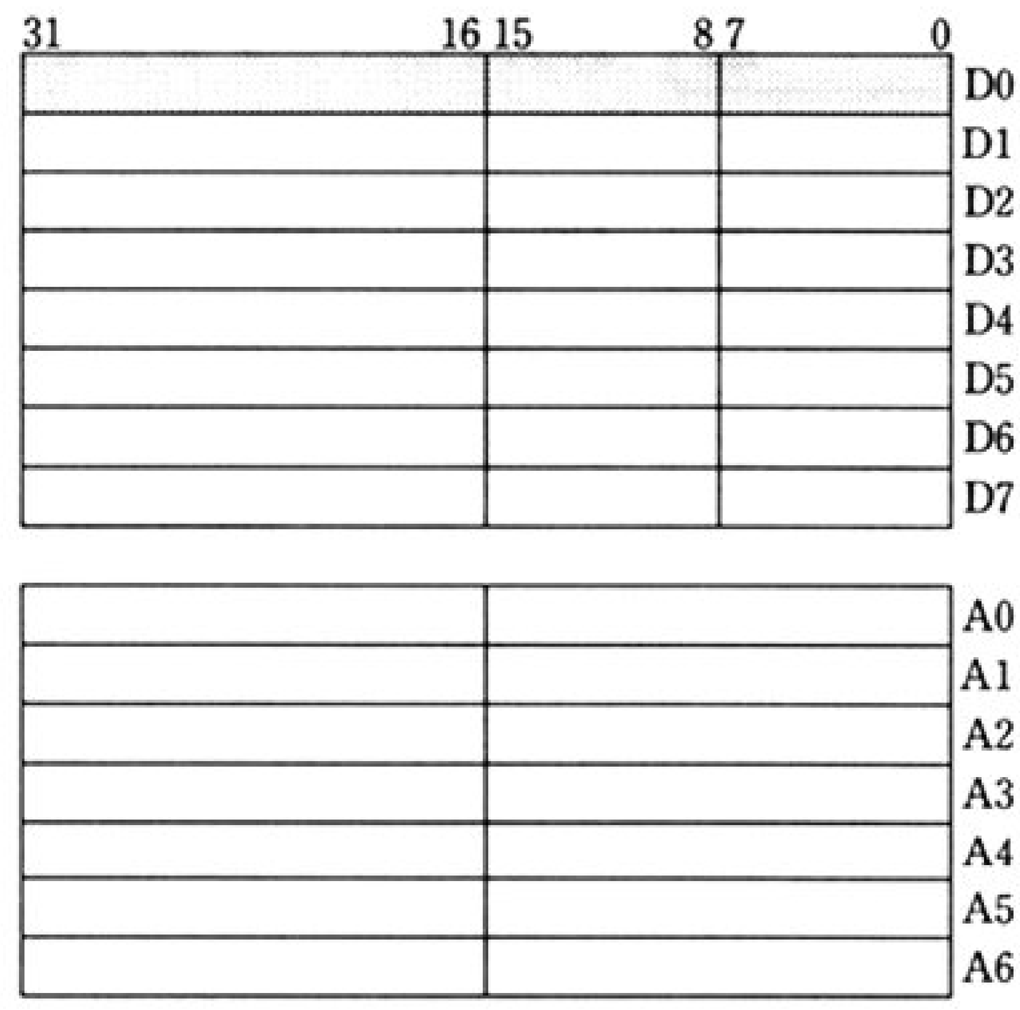
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_ROMVER,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .                       * d0.l に ROM バージョンが返ってきます
        .                       * 例 d0.l = $10870507
        .end                    * (1987年5月7日 ver1.0)
```

G_CLR_ON

● G_CLR_ON　　　IOCS コール番号：$90



機　　能　グラフィック画面をクリアして表示モードにする。

入　　力　D0. L　　$90　　IOCS コール番号

リターン　なし（D0. L の値は保証されません）

解　　説　グラフィック画面をクリアして表示モードにします。
グラフィックパレットは、標準に戻ります。
CRTMOD($10)だけでは表示されないので、このコールで表示します。

プログラム例

```
*********************************************
* G_CLR_ON                               $90
*********************************************

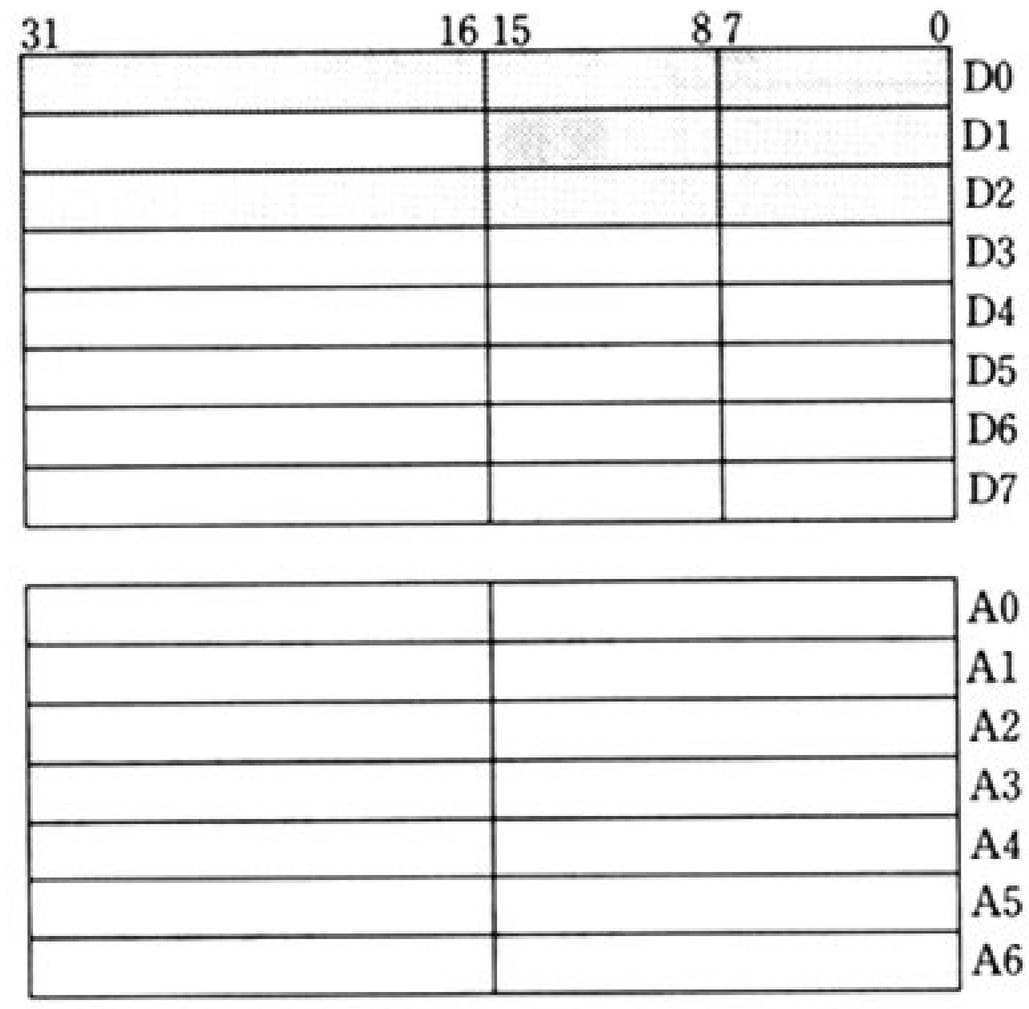
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_G_CLR_ON,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

GPALET

● GPALET　　IOCS コール番号：$94



機　能　グラフィックパレット指定

入　力

D0. L　$94　IOCS コール番号
D1. W　パレットコード
D2. L　カラーコード情報

リターン

D2. L　=　−1：
　D0. L　現在のカラーコード
D2. L　≠　−1：
　D0. L　不定

解　説　グラフィックのパレットを指定します。

入力の D1. W は、パレットコードです。
内容は次の通りです。

16 色モード時 ……0〜15
256 色モード時……0〜255
65536 色モード時… 0〜65535

入力の D2. L は、カラーコード情報です。
0〜65535 のカラーコードを指定します。
−1 を指定した場合は、現在のカラーコードを返します。

プログラム例

```
*********************************************
* GPALET                                 $94
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

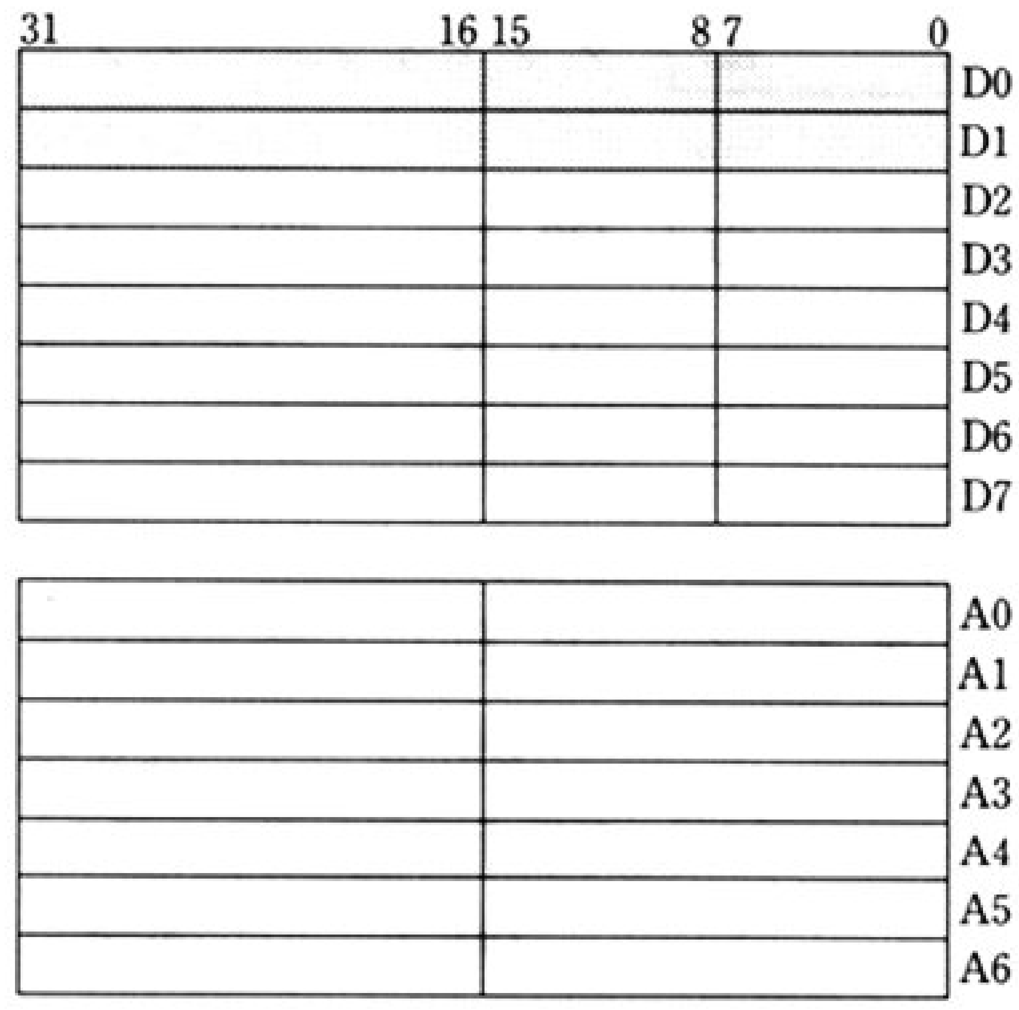
        .text
        .
        .
        moveq.l #_GPALET,d0     * IOCSコール番号をセット
        move.w  #1,d1           * パレット 1 に
        move.l  #7,d2           * カラーコード 7 をセットします
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

SFTJIS

● SFTJIS　　IOCS コール番号：$A0

機　能

シフト JIS コードを JIS 漢字コードに変換



入　力

D0. L　$A0　IOCS コール番号
D1. W　シフト JIS コード

リターン

D0. L　JIS漢字コード（上位ワードが$FFFF の場合、シフト JIS コードが不正）
D1. L　JIS 漢字コード（D0. L と同じ値）

変換した場合

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0. L	$0000															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
JIS 漢字コード															

エラーの場合

	D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
D0. L	$FFFF															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
$2228('※')															

解　説

シフト JIS コードを、JIS 漢字コードに変換します。
入力の D1. W は、シフト JIS コードです。

プログラム例

```
*******************************************
* SFTJIS                              $A0
*******************************************

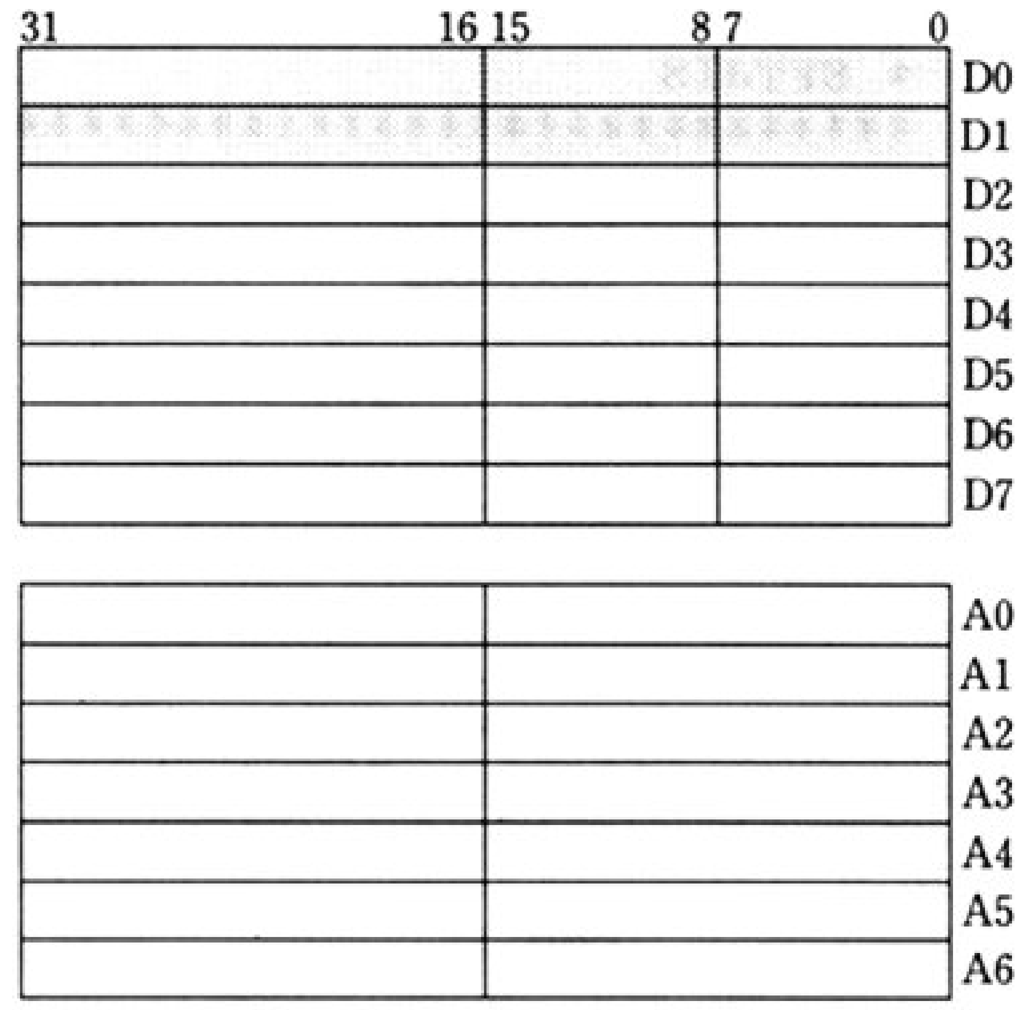
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SFTJIS,d0     * IOCSコール番号をセット
        move.w  #$8dca,d1       * '彩'のシフト JIS コード $8dca
        trap    #iocs           * IOCSコール
        .                       * d0 に '彩' の JIS コード $3a4c が返る
        .
        .end
```

JISSFT

● JISSFT　　IOCS コール番号：$A1



機　能　JIS 漢字コードをシフト JIS コードに変換

入　力　D0. L　$A1　IOCS コール番号

D1. W　JIS 漢字コード

リターン　D0. L　シフト JIS コード (上位ワードが$FFFF の場合、JIS 漢字コードが不正)

D1. L　シフト JIS コード (D0. L と同じ値)

変換した場合

	D31–D16
D0. L	$0000

	D15–D00
	シフト JIS コード

エラーの場合

	D31–D16
D0. L	$FFFF

	D15–D00
	$81A6('※')

解　説　JIS 漢字コードを、シフト JIS コードに変換します。

入力の D1. W は、JIS 漢字コードです。

プログラム例

```
********************************************
* JISSFT                                $A1
********************************************

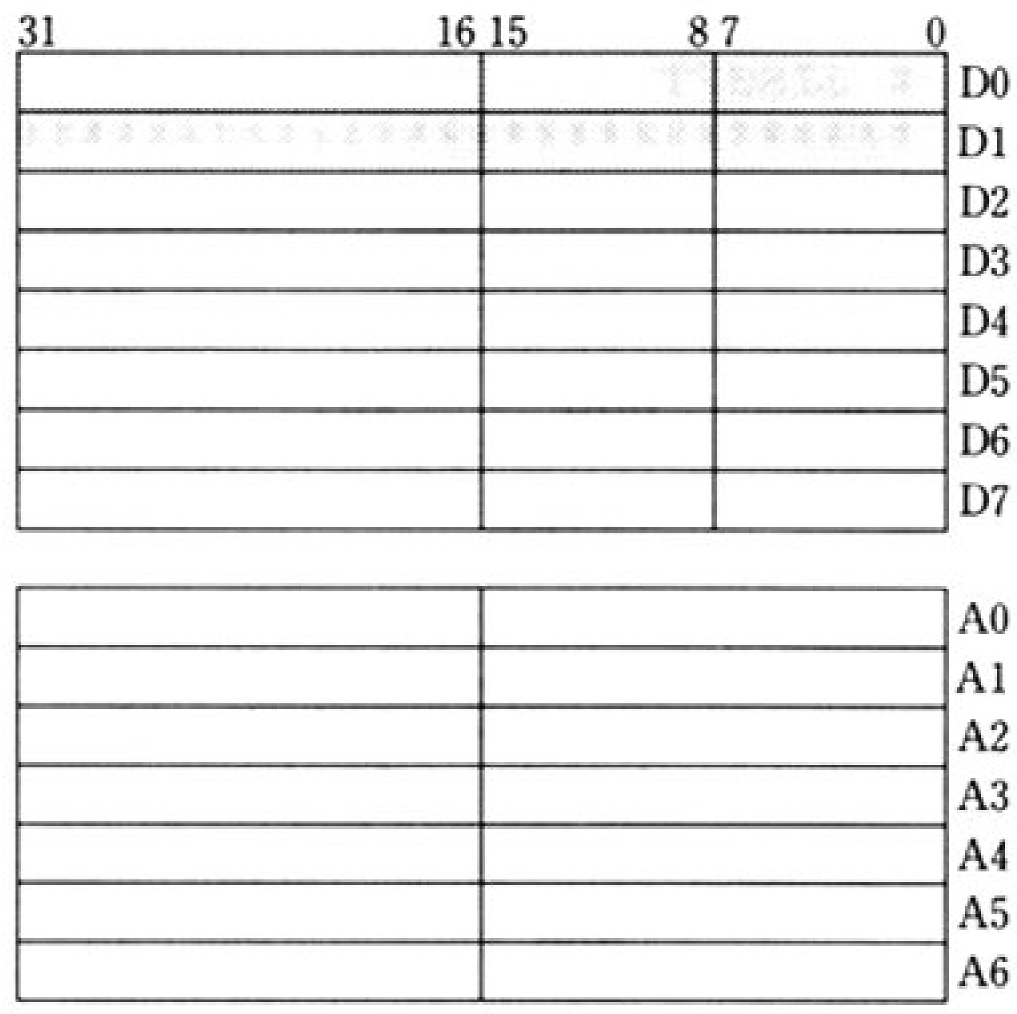
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_JISSFT,d0     * IOCSコール番号をセット
        move.w  #$3a4c,d1       * '彩'の JIS コード $3a4c
        trap    #iocs           * IOCSコール
        .                       * d0 に '彩' のシフト JIS コード $8dca
        .
        .end
```

AKCONV

● AKCONV　　IOCS コール番号：$A2



機　　能　ANK コードを全角シフト JIS コードに変換する

入　　力　D0. L　$A2　IOCS コール番号
D1. L　ANK コードと変換情報

リターン　D0. L　シフト JIS コード（上位ワードが$FFFF の場合、ANK コードが不正)

変換した場合

	D31 D30 D29 D28 D27 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16
D0. L	$0000
	D15 D14 D13 D12 D11 D10 D09 D08 D07 D06 D05 D04 D03 D02 D01 D00
	シフト JIS コード

エラーの場合

	D31 D30 D29 D28 D27 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16
D0. L	$FFFF
	D15 D14 D13 D12 D11 D10 D09 D08 D07 D06 D05 D04 D03 D02 D01 D00
	$81A6('※')

解　　説　ANK コードを、全角シフト JIS コードに変換します。
入力の D1. L は、ANK コードと変換情報です。
内容は次の通りです。

ビット 31〜16：　ひらがな、カタカナどちらに変換するかを決める
0……ひらがな
1……カタカナ
ビット 15〜 0：　ANK コード($20〜$7E または$A1〜$DF の範囲でなければならない)

プログラム例

```
********************************************
* AKCONV                                 $A2
********************************************

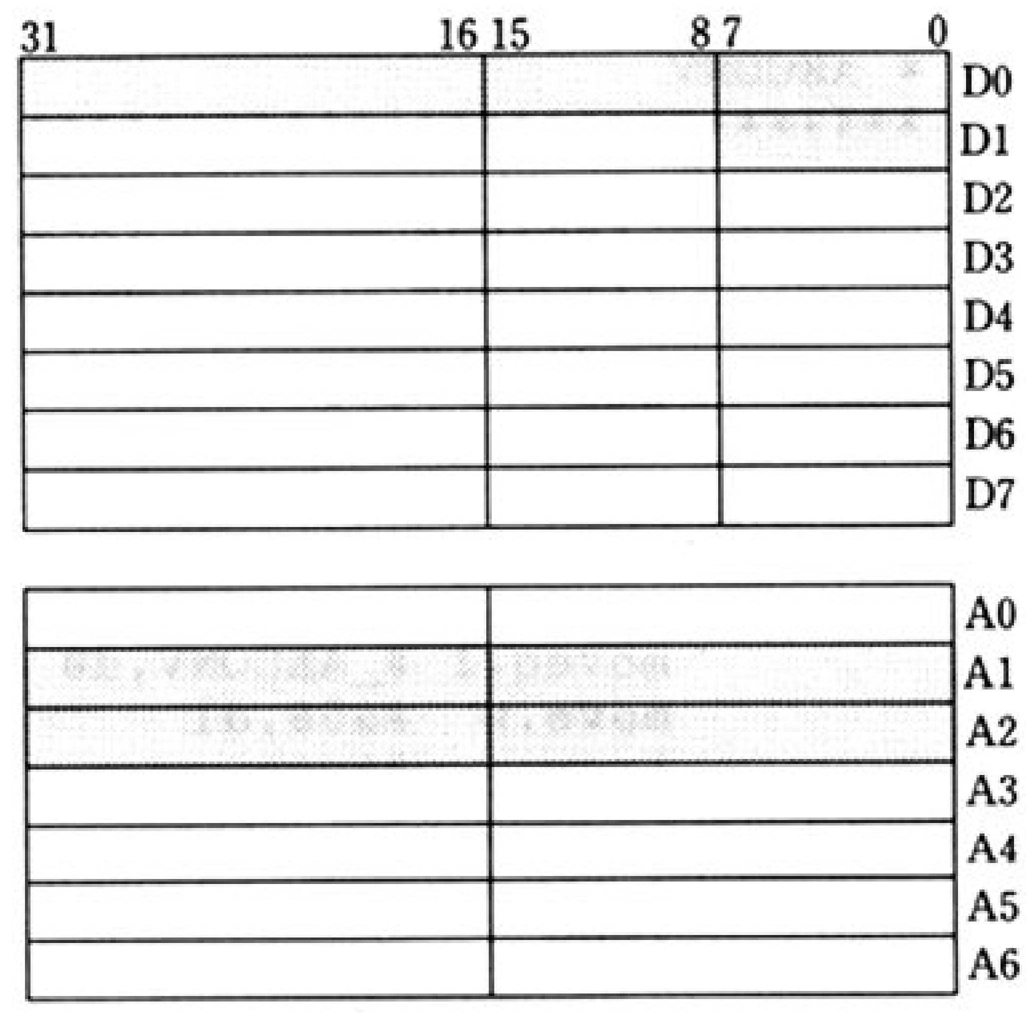
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_AKCONV,d0      * IOCSコール番号をセット
        move.w  #$58,d1          * 'X' の ASCII コード $58
        trap    #iocs            * IOCSコール
        .                        * d0 に 'X' のシフトJISコード $8277
        .
        .end
```

RMACNV

● RMACNV　　IOCSコール番号：$A3



機　　能　ローマ字かな変換

入　　力

D0. L	$A3	IOCSコール番号
D1. B	ローマ字の文字	
A1. L	変換用の作業領域の先頭アドレス	
A2. L	かな文字列（戻り値）を入れる領域の先頭アドレス	

リターン　D0. L　　ステータス

D0. L ＝ 0…………………変換途中で、返せるかな文字列がまだない
変換用の作業領域に変換途中の文字が残っているので、それを破壊してはいけません。

D0. L ＝ −1………………変換不可能な文字が指定された
作業領域の先頭のバイトは、0にされます。

D0. L ＝ 上記以外 ………変換文字数
変換用の作業領域に変換途中の文字が残っている可能性があるので、それを破壊してはいけません。
たとえばSAKKA（サッカ）を変換する場合、4文字目のKを与えた段階で“ッ”が返ります（D0. Lは1になる）が、作業領域には4文字目のKが残っている場合です。

解　　説　ローマ字かな変換を行います。
入力のD1. Bは、ローマ字の文字です。
ローマ字用のアルファベットは大文字を使用し、促音（っ）や拗音（ゃゅょ）などの“小さな”かなは、ローマ字の母音文字にアルファベットの小文字を用いることで変換されます。
つまり“ツ”はTU、“ッ”はTuで表現します。
入力のA1. Lは、変換用の作業領域の先頭アドレスです。
はじめて変換を行う場合には、作業領域の先頭のバイトに0を入れておかなければなりません。
変換用の作業領域をそれぞれのプログラムで指定することにより、複数のプログ

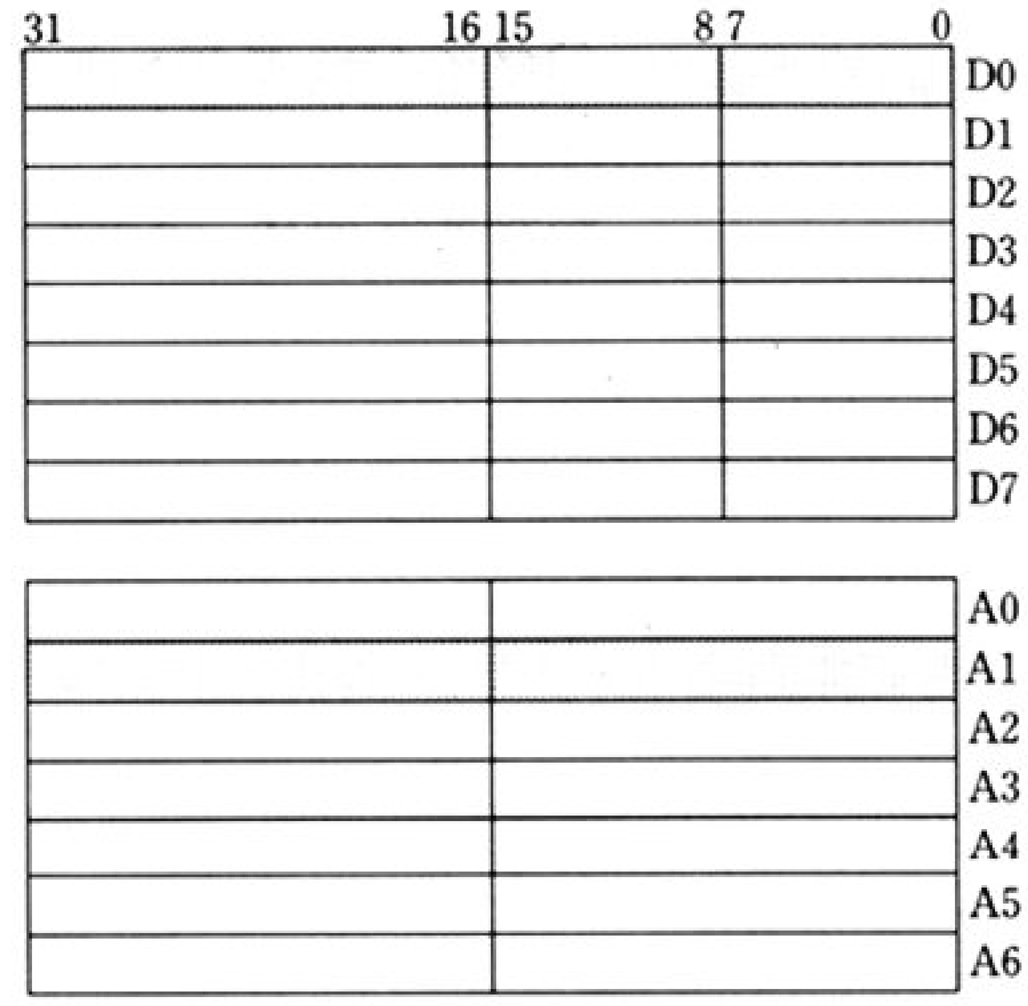
ラムが交互に変換を行っても、他のプログラムには影響を与えることがありません。

入力のA2. Lは、このコールのリターン値である、かな文字列を入れる領域の先頭アドレスです。

かな文字列は、ANKの文字列になります。

DAKJOB

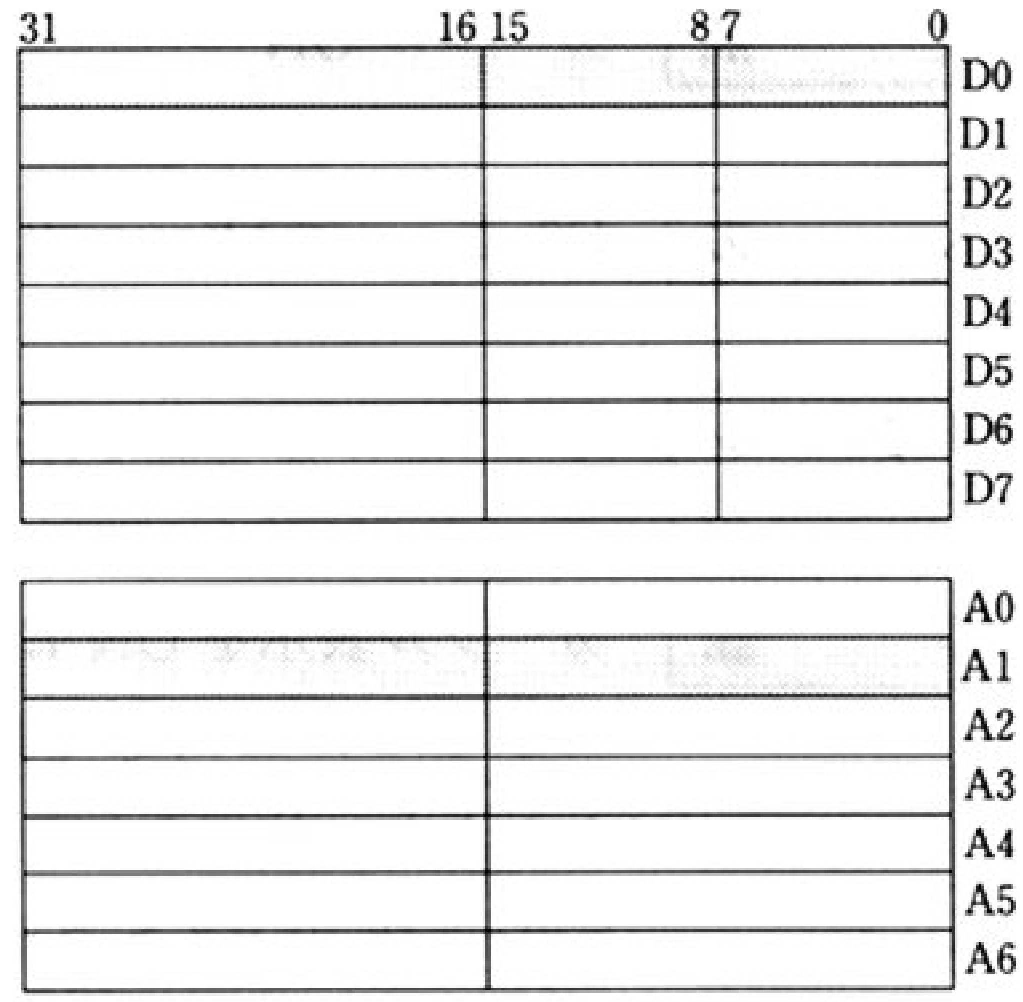
● **DAKJOB**　　**IOCS コール番号：$A4**



機　　能	全角文字列の濁点処理	
入　　力	D0. L	$A4　IOCS コール番号
	A1. L	全角文字列の終端(ヌル文字)のアドレス
リターン	D0. L	文字列の増えたバイト数 D0. L ＝0……最後の全角文字に濁点処理をした 例えば“カイセツ”+0が“カイセヅ”+0になったときなどです。 D0. L ＝2……゛(濁点)を追加した 例えば“リカイ”+0が“リカイ゛”+0になったときなどです。
	A1. L	処理後の全角文字列の終端(ヌル文字)のアドレス このアドレスの示す内容は、ヌル文字(0)になっています。
解　　説	全角文字列の濁点処理を行います。 入力の A1. L は、全角文字列の終端(ヌル文字)のアドレスです。 そのアドレスの示す内容は、ヌル文字(0)でなければなりません。	

HANJOB

● HANJOB　　IOCS コール番号：$A5



機　能　全角文字列の半濁点処理

入　力　D0. L　$A5　IOCS コール番号
A1. L　全角文字列の終端(ヌル文字)のアドレス

リターン　D0. L　文字列の増えたバイト数
D0. L　＝0……最後の全角文字に半濁点処理をした。例えば"ワイフ"+0が"ワイプ"+0になったときなどです。
D0. L　＝2……°（半濁点）を追加した
例えば"ズーム"＋0が"ズーム°"＋0になったときなどです。
A1. L　処理後の全角文字列の終端(ヌル文字)のアドレス
このアドレスの示す内容は、ヌル文字(0)になっています。

解　説　全角文字列の半濁点処理をします。
入力の A1. L は、全角文字列の終端(ヌル文字)のアドレスです。
そのアドレスの示す内容は、ヌル文字(0)でなければなりません。

OS_CURON

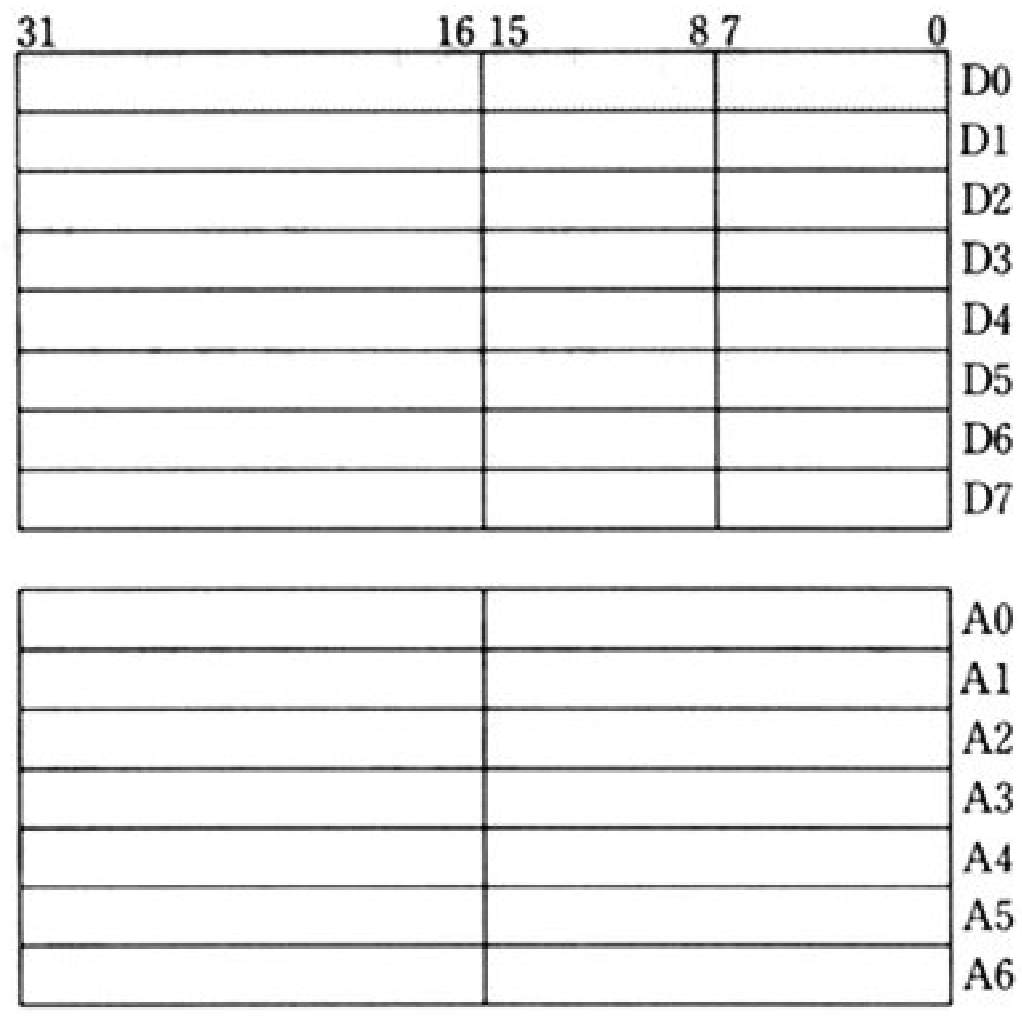
● OS_CURON　　　IOCS コール番号：$AE

機　能　カーソル ON

入　力　D0. L　　$AE　　IOCS コール番号

リターン　なし（D0. L の値は保証されません）

解　説　カーソル表示を ON にします。
コンソールに対して ESC［>5l を出力したときに使用されます。
$20～$2F の IOCS コールに対して有効です。



プログラム例

```
*******************************************
* OS_CURON                              $AE
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

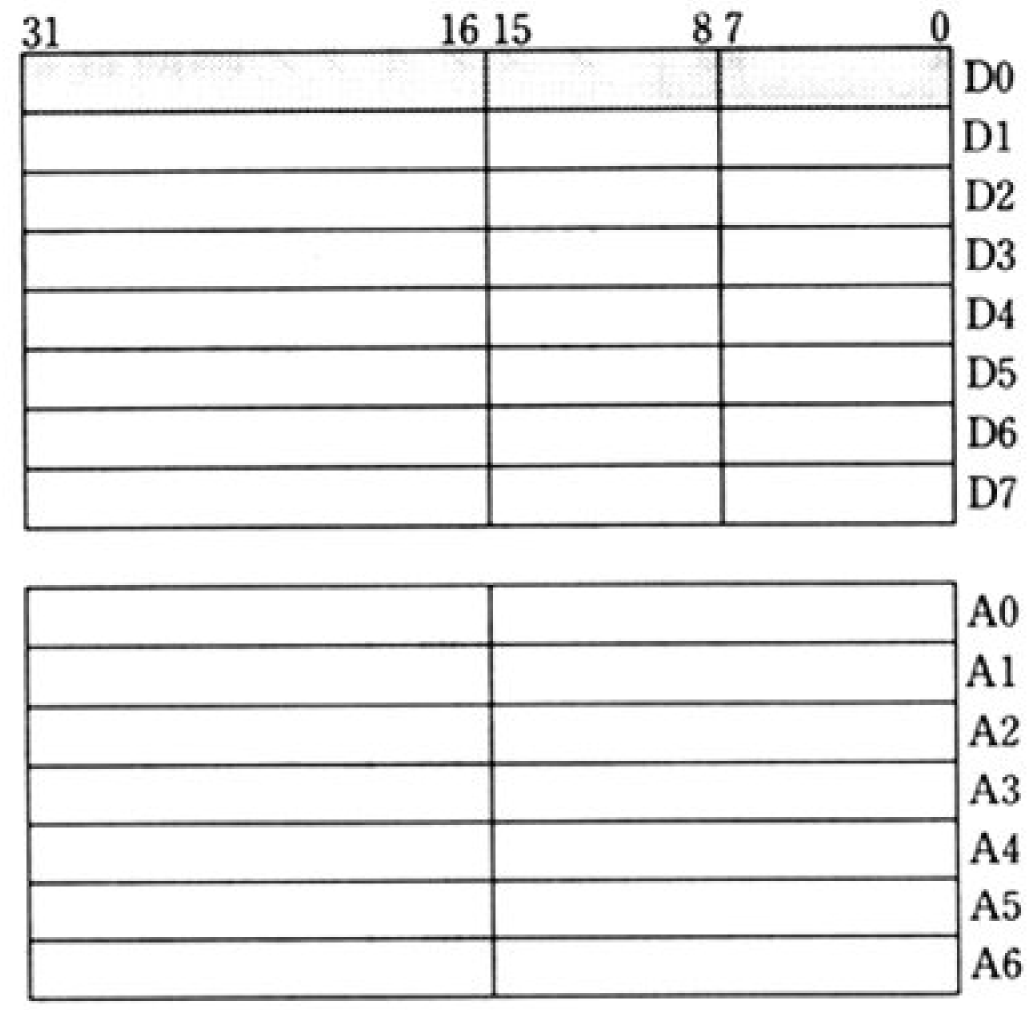
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OS_CURON,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

OS_CUROF

● OS_CUROF　　IOCS コール番号：$AF

機　能　カーソル OFF



入　力　D0. L　$AF　IOCS コール番号

リターン　なし（D0. L の値は保証されません）

解　説　カーソル表示を OFF にします。
コンソールに対して ESC [>5h を出力したときに使用されます。
$20〜$2F の IOCS コールに対して有効です。

プログラム例

```
********************************************
* OS_CUROF                              $AF
********************************************

        .include        iocscall.mac
        .include        doscall.mac

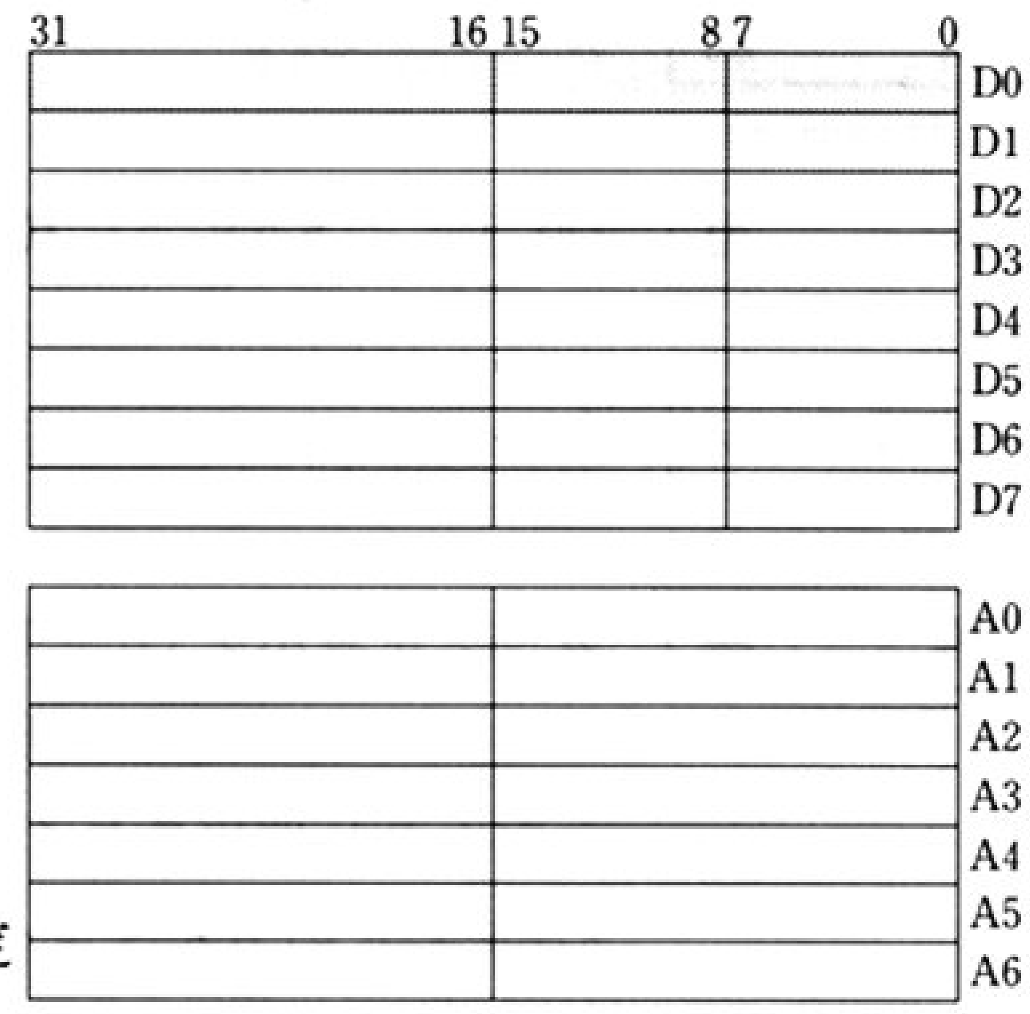
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_OS_CUROF,d0   * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

APAGE

● APAGE　　IOCS コール番号：$B1

機　能　グラフィック画面書き込みページ設定



入　力　D0. L　$B1　IOCS コール番号

D1. B　書き込みページ指定

リターン　D1. B ≠ −1：

D0. L ＝　0……正常終了

D0. L ＝ −1……グラフィックは使用不可

D0. L ＝ −2……書き込みページ指定が指定外

D0. L ＝ −3……指定されたページは現在のモードでは設定不可

D1. B ＝ −1：

D0. L 現在の書き込みページ (0～3)

解　説　グラフィック画面の、書き込みページの設定をします。

入力の D1. B は、書き込みページ指定です。

内容は次の通りです。

D1. B　＝　0………0 ページを設定

D1. B　＝　1………1 ページを設定

D1. B　＝　2………2 ページを設定

D1. B　＝　3………3 ページを設定

D1. B　＝　−1……現在の書き込みページを調べます

プログラム例

```
*******************************************
* APAGE                                $B1
*******************************************

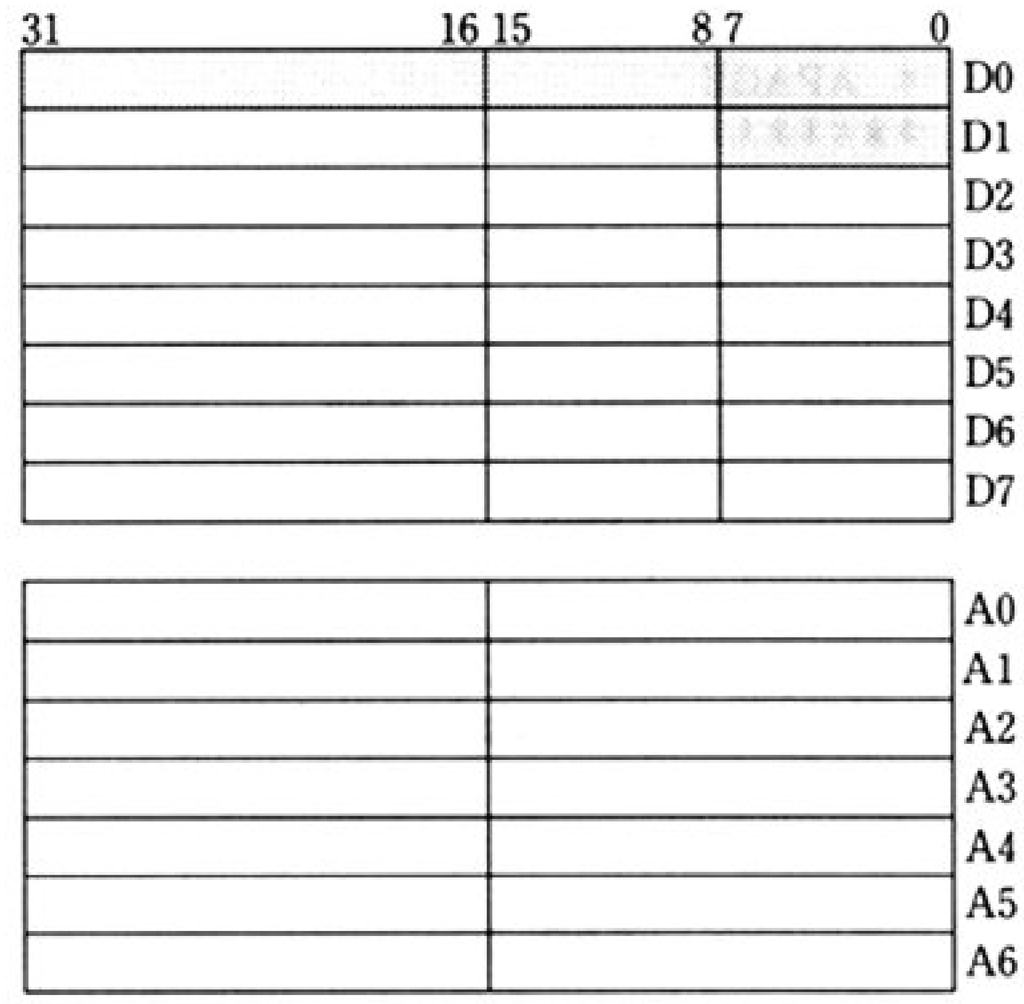
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_APAGE,d0       * IOCSコール番号をセット
        move.b  #0,d1            * アクティブページ 0
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

VPAGE

● VPAGE　　IOCS コール番号：$B2



機　能　グラフィック画面表示ページ設定

入　力　D0. L　$B2　IOCS コール番号
D1. B　表示ページの指定

リターン　D0. L　終了状態
D0. L ＝ 0 ……正常終了
D0. L ＝ −1 …グラフィックは使用不可
D0. L ＝ −2 …ページ引数が指定外
D0. L ＝ −3 …指定されたページは現在のモードでは設定不可

解　説　グラフィック画面の表示を設定します。
入力の D1. B は、表示ページの指定です。
内容は次の通りです。
ビット 3 (PG3)……3 ページ
ビット 2 (PG2)……2 ページ
ビット 1 (PG1)……1 ページ
ビット 0 (PG0)……0 ページ
ビットが 1 のとき表示され、0 のとき表示されません。

プログラム例

```
*********************************************
* VPAGE                                  $B2
*********************************************

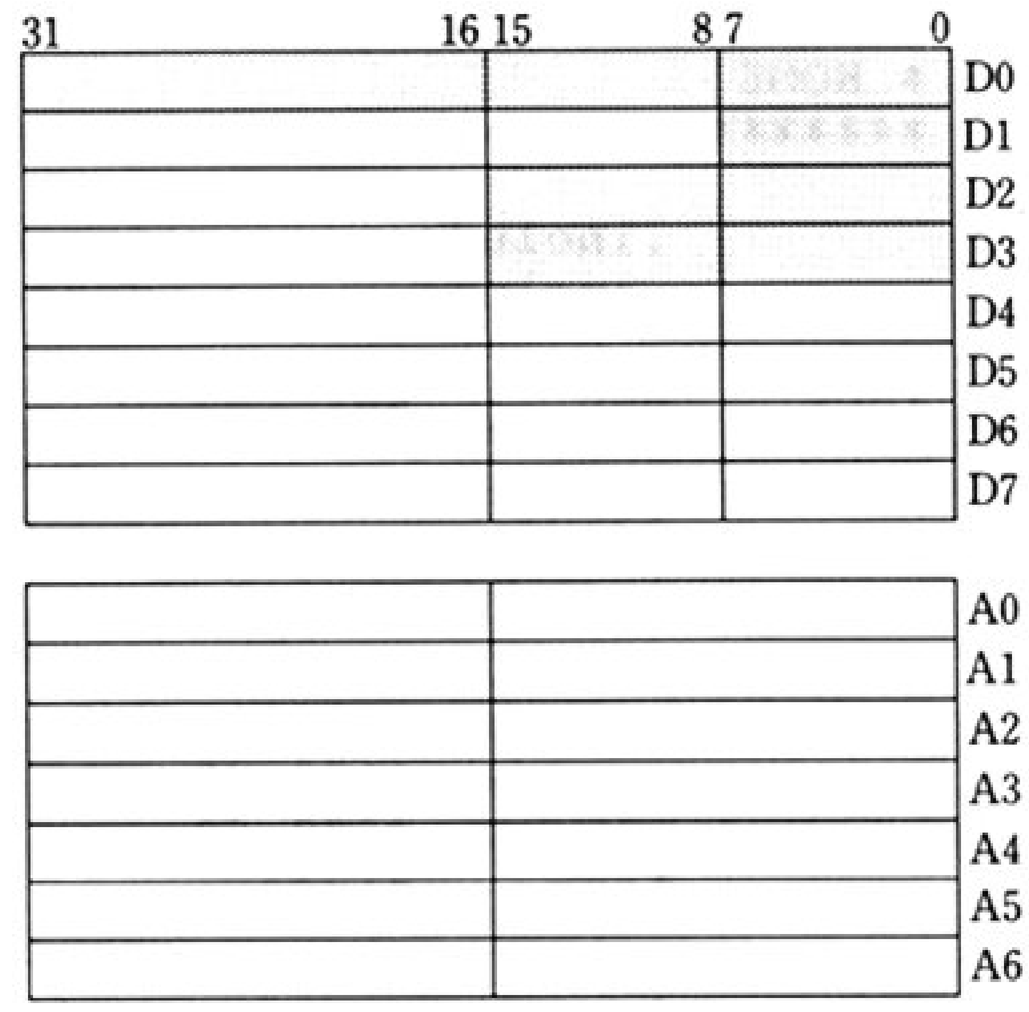
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_VPAGE,d0      * IOCSコール番号をセット
        move.b  #%0001,d1       * 表示ページ 0 ページ
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

HOME

● HOME　　IOCS コール番号：$B3



機　能　グラフィック画面の表示位置の設定

入　力

D0. L	$B3	IOCS コール番号
D1. B	ページ指定	
D2. W	X 座標	
D3. W	Y 座標	

リターン　D0. L　　終了状態

D0. L　=　0 ……正常終了
D0. L　=　−1 ……グラフィックは使用不可
D0. L　=　−2 ……ページ引数が指定外
D0. L　=　−3 ……指定されたページは現在のモードでは設定不可

解　説　グラフィック画面の表示位置を設定します。
指定座標を画面の左上に設定します。
入力の D1. B はページ指定です。
値が 0 の場合、現在のモードで有効なページがすべて変更されます。
0 以外の場合の内容は次の通りです。

ビット 3 (PG3)……3 ページ
ビット 2 (PG2)……2 ページ
ビット 1 (PG1)……1ページ
ビット 0 (PG0)……0 ページ

ビットが 1 のとき変更され、0 のとき変更されません。
入力の D2. W は、X 座標です。
値は 0 から 1023 の範囲です。
入力の D3. W は、Y 座標です。
値は 0 から 1023 の範囲です。

プログラム例

```
******************************************
* HOME                               $B3
******************************************

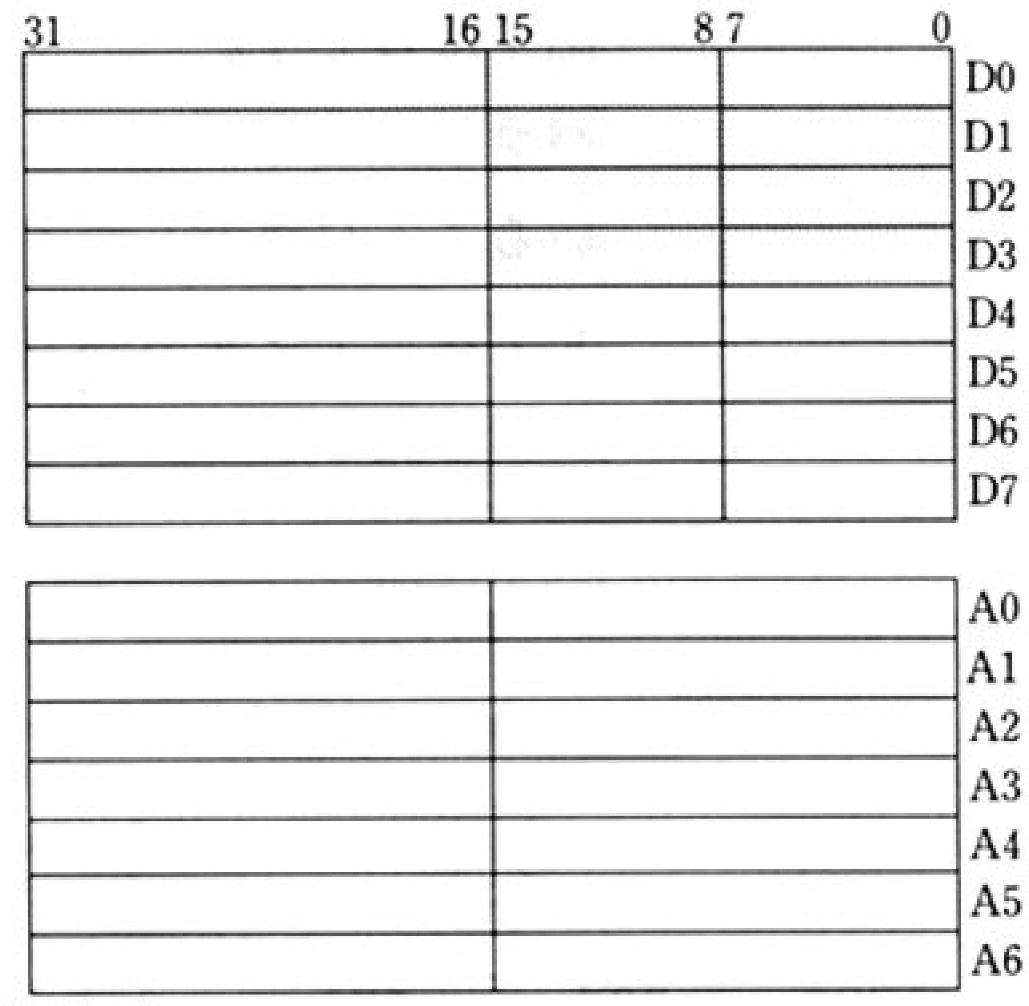
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_HOME,d0       * IOCSコール番号をセット
        move.b  #%0001,d1       * 0 ページの表示位置
        move.w  #100,d2         * CRTの左が仮想画面の 100
        move.w  #200,d3         * CRTの上が仮想画面の 200
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

WINDOW

● WINDOW　　IOCS コール番号：$B4



機　能　グラフィック画面のウィンドウの設定

入　力
D0. L　　$B4　　IOCS コール番号
D1. W　　ウィンドウ左上 X 座標
D2. W　　ウィンドウ左上 Y 座標
D3. W　　ウィンドウ右下 X 座標
D4. W　　ウィンドウ右下 Y 座標

リターン　D0. L　　終了状態
D0. L　＝　0……正常終了
D0. L　＝　−1……グラフィックは使用不可
D0. L　＝　−2……引数が指定通りになっていません

解　説　グラフィック画面のウィンドウを設定します。
入力の D1. W、D2. W、D3. W、D4. W は、それぞれウィンドウ左上 X 座標、ウィンドウ左上 Y 座標、ウィンドウ右下 X 座標、ウィンドウ右下 Y 座標です。
値は 0 から 1023 の範囲ですが、D2. W≦D4. W かつ D1. W≦D3. W でなければなりません。
ウィンドウ座標は、表示画面の座標です。
LINE($B8) などを実行したときにウィンドウ内だけ描画されます（クリッピング)。

プログラム例 ラインを2本引き、ウィンドウの変化によるクリッピングの動作を確認します。

メインプログラムは、パラメータをスタックにプッシュしてサブルーチンを呼び出すだけです。

パラメータをプッシュする順番が、ベーシックのパラメータの順番と逆になっていることに注意してください。

サブルーチンから帰ってきたらプッシュしたパラメータのバイト数に応じてスタックを戻します。

```
********************************************
* WINDOW                                 $B4
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
MAIN:
        bsr     SCREEN          * screen
        move.w  #1,-(sp)        * vpage(1)
        bsr     VPAGE           *
        addq.l  #2,sp           *
        bsr     WIPE            * wipe()
        move.w  #511,-(sp)      * window(0, 0, 767, 511)
        move.w  #767,-(sp)      * サイズ 2 バイトのパラメータを 4 つ
        move.w  #0,-(sp)        * プッシュします
        move.w  #0,-(sp)        *
        bsr     WINDOW          * サブルーチンを呼びます
        addq.l  #8,sp           * スタックを元に戻します(2×4 = 8byte)
        move.w  #$ffff,-(sp)    * line(0, 0, 767, 511, 15, &hffff)
        move.w  #15,-(sp)       *
        move.w  #511,-(sp)      *
        move.w  #767,-(sp)      *
        move.w  #0,-(sp)        *
        move.w  #0,-(sp)        *
        bsr     LINE            *
        lea     12(sp),sp       *
        move.w  #200,-(sp)      * window(0, 0, 200, 200)
        move.w  #200,-(sp)      *
        move.w  #0,-(sp)        *
        move.w  #0,-(sp)        *
        bsr     WINDOW          *
        addq.l  #8,sp           *
        move.w  #$ffff,-(sp)    * line(0, 0, 767, 256, 15, &hffff)
        move.w  #15,-(sp)       *
        move.w  #256,-(sp)      *
        move.w  #767,-(sp)      *
        move.w  #0,-(sp)        *
        move.w  #0,-(sp)        *
        bsr     LINE            *
        lea     12(sp),sp       *
        dc.w    _EXIT           * end


SCREEN:
        moveq.l #_CRTMOD,d0
        move.w  #16,d1
        trap    #iocs
        moveq.l #_G_CLR_ON,d0
        trap    #iocs
        rts
```

```
VPAGE:                              *+++ View Page +++
        moveq.l #_VPAGE,d0
        move.b  5(sp),d1            * 5(sp)が最後にpushしたパラメータ(下位バイト)
        trap    #iocs               * bsr命令の戻り番地がpushされたから
        rts

WIPE:                               *+++ Wipe +++
        moveq.l #_WIPE,d0
        trap    #iocs
        rts

LINE:                               *+++ Line +++
        moveq.l #_LINE,d0
        movea.l sp,a1
        addq.l  #4,a1
        trap    #iocs
        rts

WINDOW:                             *+++ Windoiw +++
        moveq.l #_WINDOW,d0         * IOCSコール番号をセット
        move.w  4(sp),d1
        move.w  6(sp),d2
        move.w  8(sp),d3
        move.w  10(sp),d4
        trap    #iocs               * IOCSコール
        rts

        .end
```

WIPE

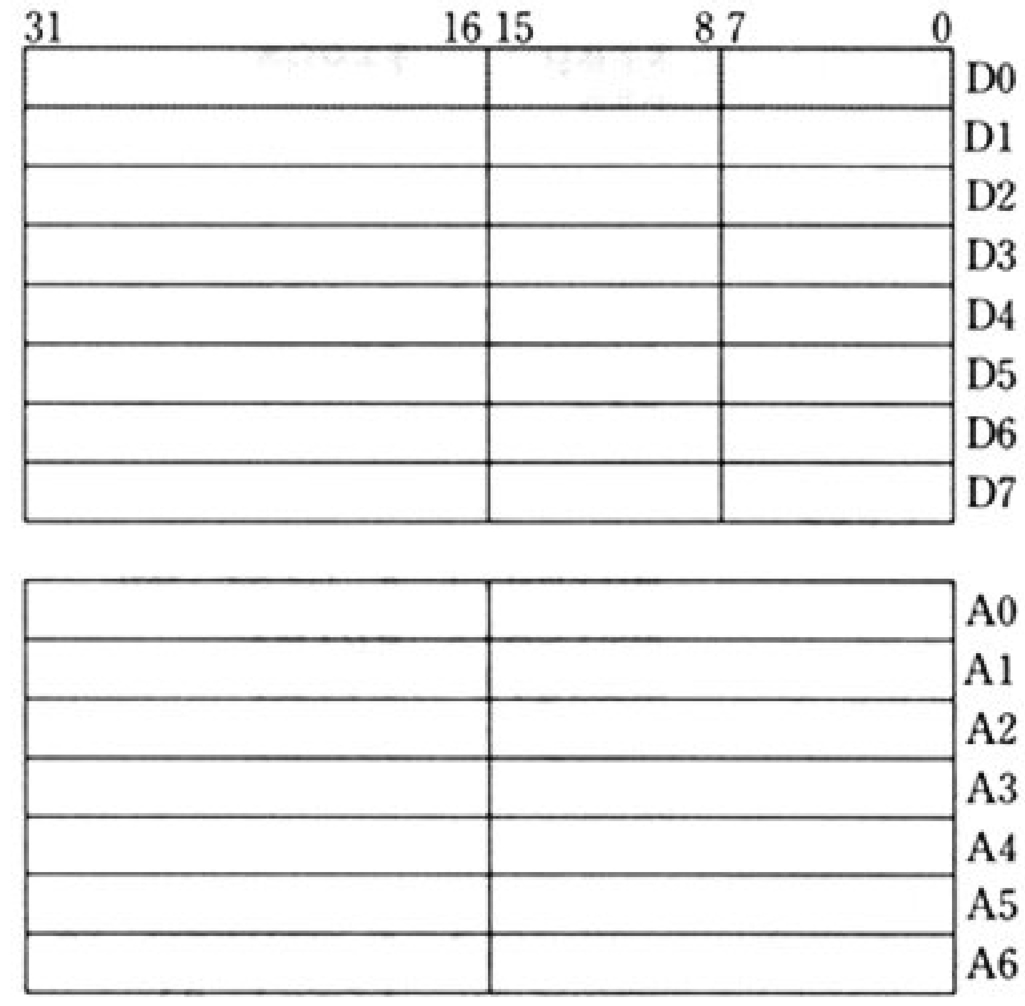
● WIPE　　IOCS コール番号：$B5

機　能　グラフィック画面のクリア

入　力　D0. L　$B5　IOCS コール番号

リターン　D0. L　終了状態
D0. L　＝　0……正常終了
D0. L　＝　−1…グラフィック画面使用不可

解　説　グラフィック画面をクリアします。



プログラム例

```
*******************************************
* WIPE                                  $B5
*******************************************

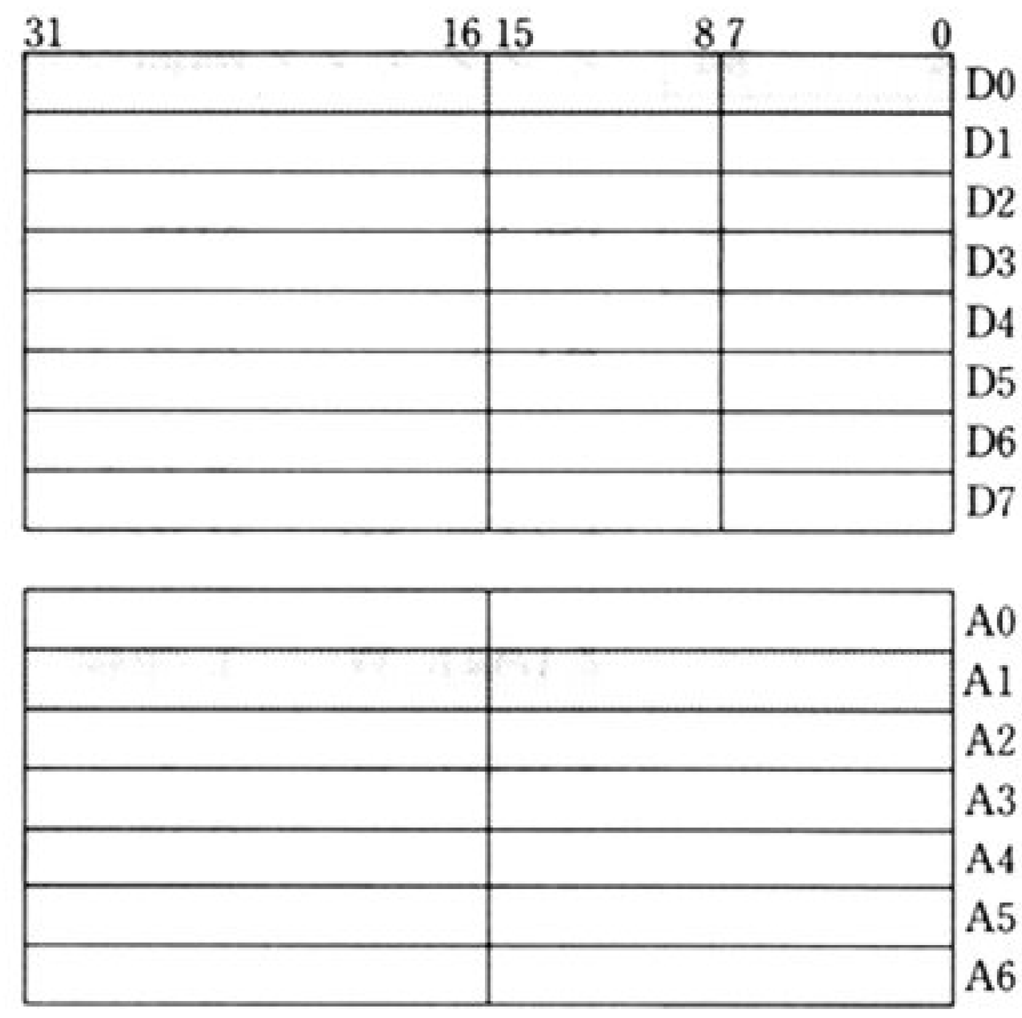
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_WIPE,d0       * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

PSET

● PSET　　IOCS コール番号：$B6



機　　能　グラフィック画面のポイントセット

入　　力　D0. L　　$B6　　IOCS コール番号

A1. L　　パラメータが格納されているアドレスの先頭

0 (A1). W　　X 座標

2 (A1). W　　Y 座標

4 (A1). W　　パレットコード

リターン　D0. L　　終了状態

D0. L　＝　0………正常終了

D0. L　＝　−1……グラフィック画面使用不可

解　　説　グラフィック画面に、ポイントセットを行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

0 (A1). W、2 (A1). W、4 (A1). Wはそれぞれ、X座標、Y座標、パレットコードです。

プログラム例

```
*******************************************
* PSET                                  $B6
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

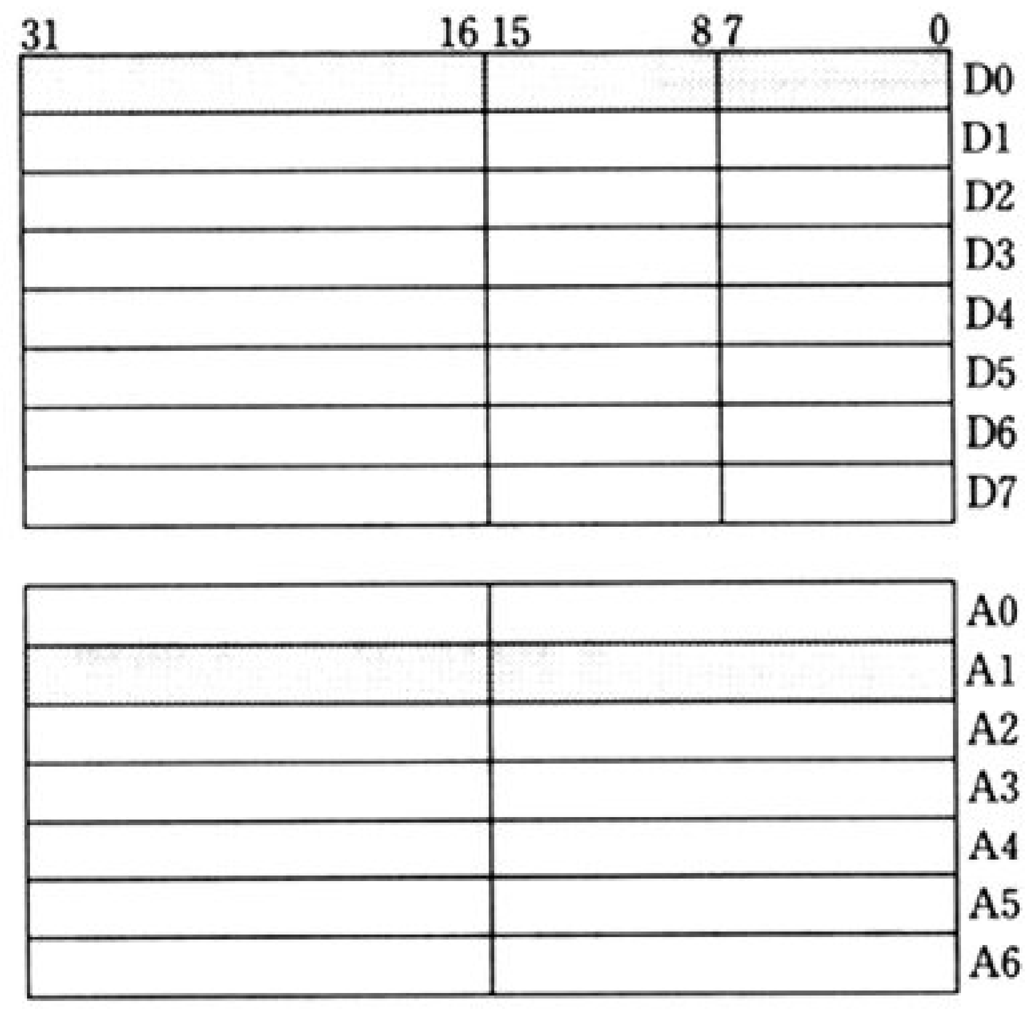
iocs    equ     $0f

        .text
        .
        .
        move.w  #01,-(sp)       * パレットコード
        move.w  #256,-(sp)      * Y 座標
        move.w  #100,-(sp)      * X 座標
        bsr     PSET
        addq.l  #6,sp
        .
        .

PSET:   moveq.l #_PSET,d0       * IOCSコール番号をセット
        movea.l sp,a1
        addq.l  #4,a1           * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

POINT

● POINT　　IOCS コール番号：$B7



機　　能　グラフィック画面のポイントゲット

入　　力

D0. L	$B7	IOCS コール番号
A1. L	パラメータが格納されているアドレスの先頭	
0 (A1). W	X 座標	
2 (A1). W	Y 座標	
4 (A1). W	パレットコードのリターン値用の領域	

リターン　D0. L　　終了状態

D0. L　＝ 0………正常終了

D0. L　＝ −1……グラフィック画面使用不可

解　　説　グラフィック画面のポイントゲットを行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

0 (A1). W、2 (A1). W、4 (A1). W はそれぞれ、X 座標、Y 座標、パレットコードのリターン値用の領域（空けておく）です。

パレットコードの 4(A1). W には、リターン値の D0. L が 0（正常終了）のとき、パレットコードが返ります。

プログラム例

```
********************************************
* POINT                                 $B7
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

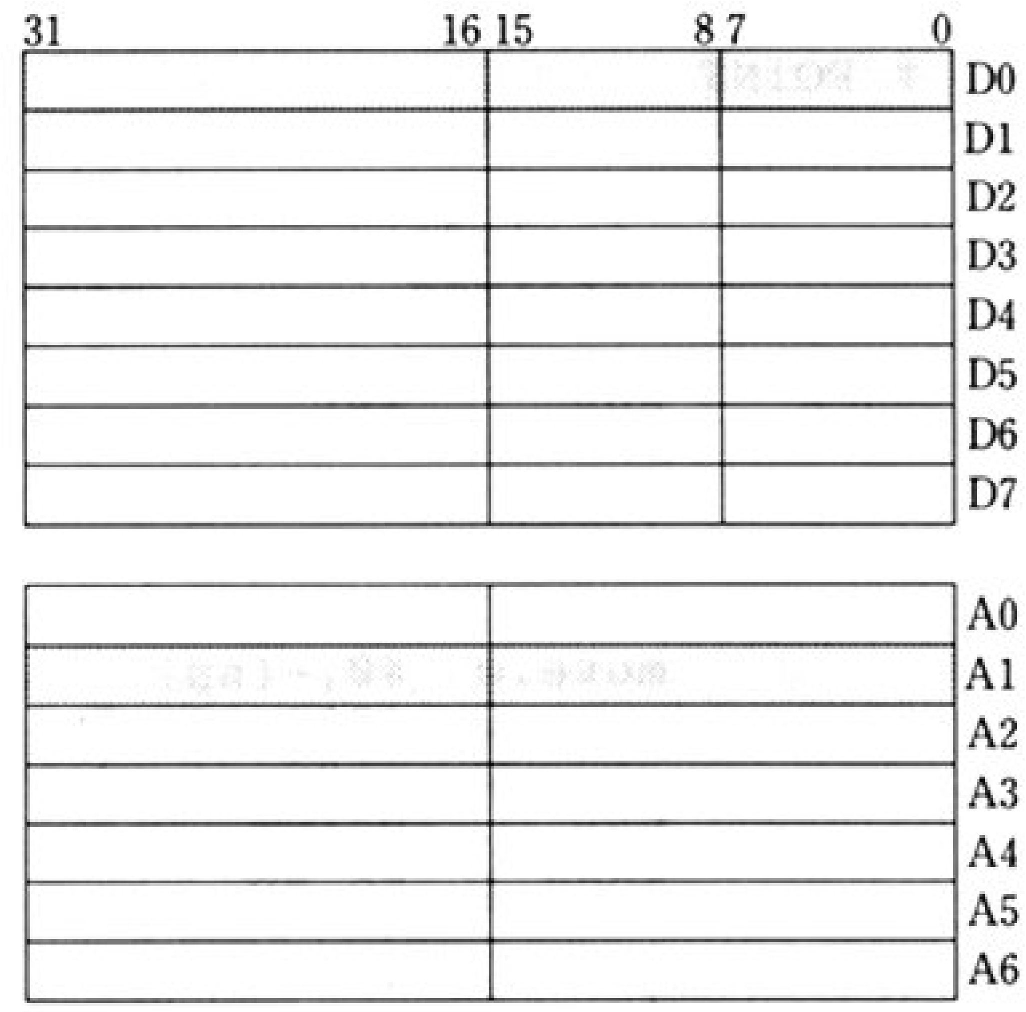
        .text
        .
        .
        move.w  #0,-(sp)        * パレットコードが帰ってくる領域を確保
        move.w  #256,-(sp)      * Y 座標 = 256
        move.w  #100,-(sp)      * X 座標 = 100
        bsr     POINT
        addq.l  #6,sp           * -2(sp) にパレットコードが返ってきます
        .
        .

POINT:  moveq.l #_POINT,d0      * IOCSコール番号をセット
        movea.l sp,a1
        addq.l  #4,a1
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

LINE

● LINE　　IOCS コール番号：$B8

機　能	グラフィック画面のライン	
入　力	D0. L	$B8　IOCS コール番号
	A1. L	パラメータが格納されているアドレスの先頭
	0 (A1). W	始点 X 座標
	2 (A1). W	始点 Y 座標
	4 (A1). W	終点 X 座標
	6 (A1). W	終点 Y 座標
	8 (A1). W	パレットコード
	10 (A1). W	ラインスタイル



リターン　D0. L　終了状態

D0. L　＝ 0………正常終了

D0. L　＝ −1……グラフィック画面使用不可

解　説　グラフィック画面のライン描画を行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W はそれぞれ始点の X、Y 座標です。

入力の 4 (A1). W、6 (A1). W はそれぞれ終点の X、Y 座標です。

入力の 8 (A1). W、10 (A1). W はそれぞれパレットコード、ラインスタイルです。

プログラム例

```
******************************************
* LINE                                $B8
******************************************

        .include        iocscall.mac
        .include        doscall.mac

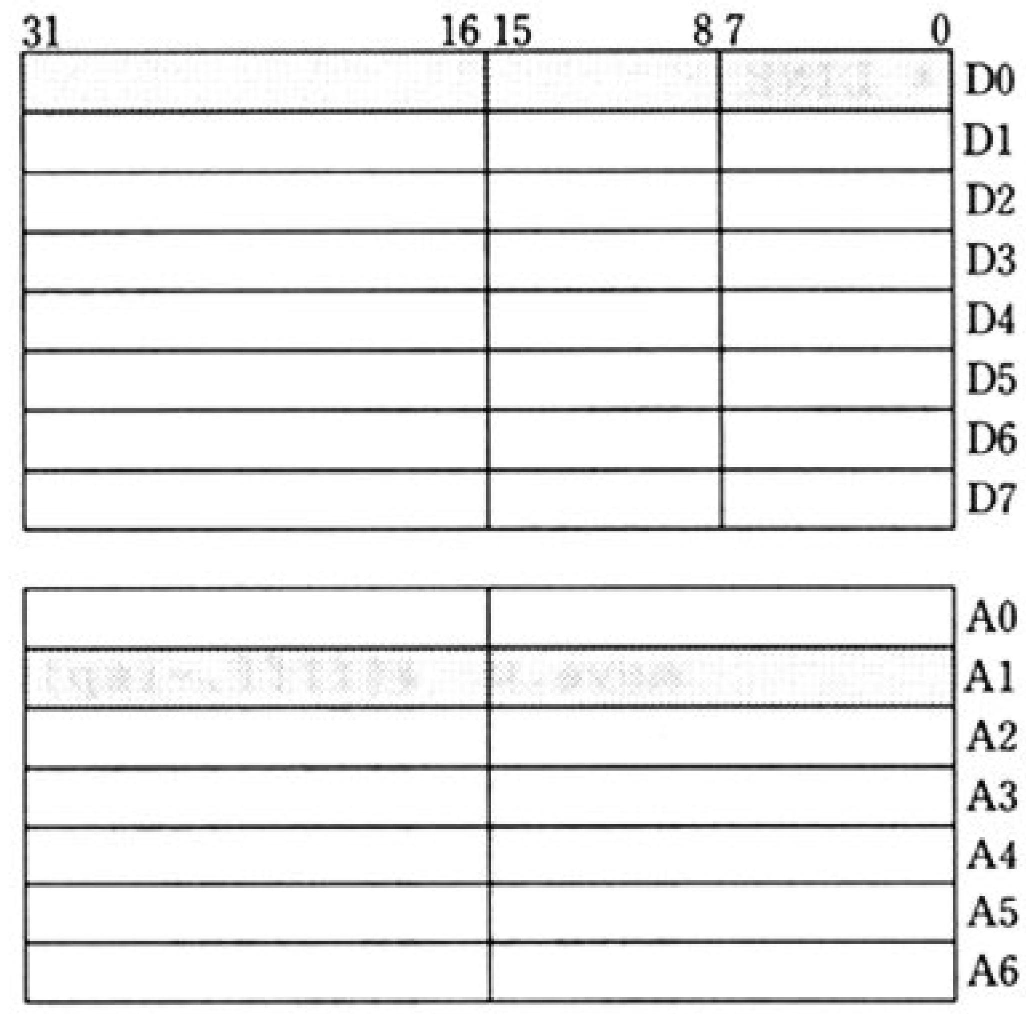
iocs    equ     $0f

        .text
        .
        .                       * line(0, 0, 767, 511, 15, &hffff)
        move.w  #$ffff,-(sp)    * ラインパターン
        move.w  #15,-(sp)       * パレットコード
        move.w  #511,-(sp)      * Y2
        move.w  #767,-(sp)      * X2
        move.w  #0,-(sp)        * Y1
        move.w  #0,-(sp)        * X1
        bsr     LINE            * サブルーチンを呼びます
        lea     12(sp),sp       * スタックを元に戻します
        .
        .

LINE:                           *+++ Line +++
        moveq.l #_LINE,d0       * IOCSコール番号をセット
        movea.l sp,a1           *
        addq.l  #4,a1           * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

BOX

● BOX　　IOCS コール番号：$B9



機　能　グラフィック画面のボックス

入　力

D0. L　$B9　IOCS コール番号

A1. L　パラメータが格納されているアドレスの先頭

0 (A1). W　始点 X 座標

2 (A1). W　始点 Y 座標

4 (A1). W　終点 X 座標

6 (A1). W　終点 Y 座標

8 (A1). W　パレットコード

10 (A1). W　ラインスタイル

リターン

D0. L　終了状態

D0. L　＝ 0………正常終了

D0. L　＝ −1……グラフィック画面使用不可

解　説

グラフィック画面のボックス描画を行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W はそれぞれ始点の X、Y 座標です。

入力の 4 (A1). W、6 (A1). W はそれぞれ終点の X、Y 座標です。

入力の 8 (A1). W、10 (A1). W はそれぞれパレットコード、ラインスタイルです。

プログラム例

```
*******************************************
* BOX                                    $B9
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

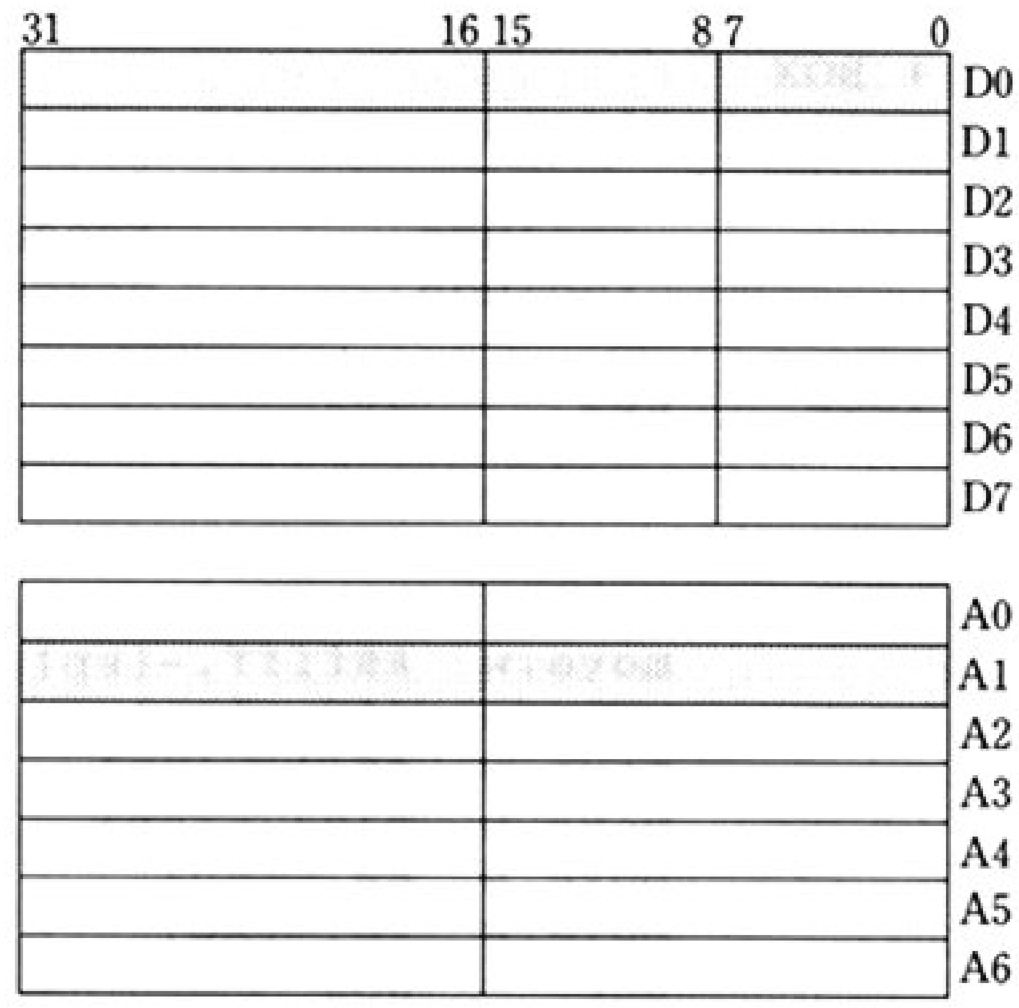
iocs    equ     $0f

        .text
        .
        .                       * box(0, 0, 767, 511, 15, &hffff)
        move.w  #$ffff,-(sp)    * ラインスタイル
        move.w  #15,-(sp)       * パレットコード
        move.w  #511,-(sp)      * Y2
        move.w  #767,-(sp)      * X2
        move.w  #0,-(sp)        * Y1
        move.w  #0,-(sp)        * X1
        bsr     BOX             * サブルーチンを呼びます
        lea     12(sp),sp       * スタックを元に戻します
        .
        .

BOX:                            *+++ Box +++
        moveq.l #_BOX,d0        * IOCSコール番号をセット
        movea.l sp,a1           *
        addq.l  #4,a1           * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

FILL

● FILL　　IOCS コール番号：$BA



機　能	グラフィック画面のボックスフル	
入　力	D0. L	$BA　IOCS コール番号
	A1. L	パラメータが格納されているアドレスの先頭
	0 (A1). W	始点 X 座標
	2 (A1). W	始点 Y 座標
	4 (A1). W	終点 X 座標
	6 (A1). W	終点 Y 座標
	8 (A1). W	パレットコード
リターン	D0. L	終了状態

D0. L ＝ 0………正常終了

D0. L ＝ −1……グラフィック画面は使用不可です

解　説

グラフィック画面のボックスフル描画を行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W はそれぞれ始点の X、Y 座標です。

入力の 4 (A1). W、6 (A1). W はそれぞれ終点の X、Y 座標です。

入力の 8 (A1). W はパレットコードです。

プログラム例

```
*******************************************
* FILL                                  $BA
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

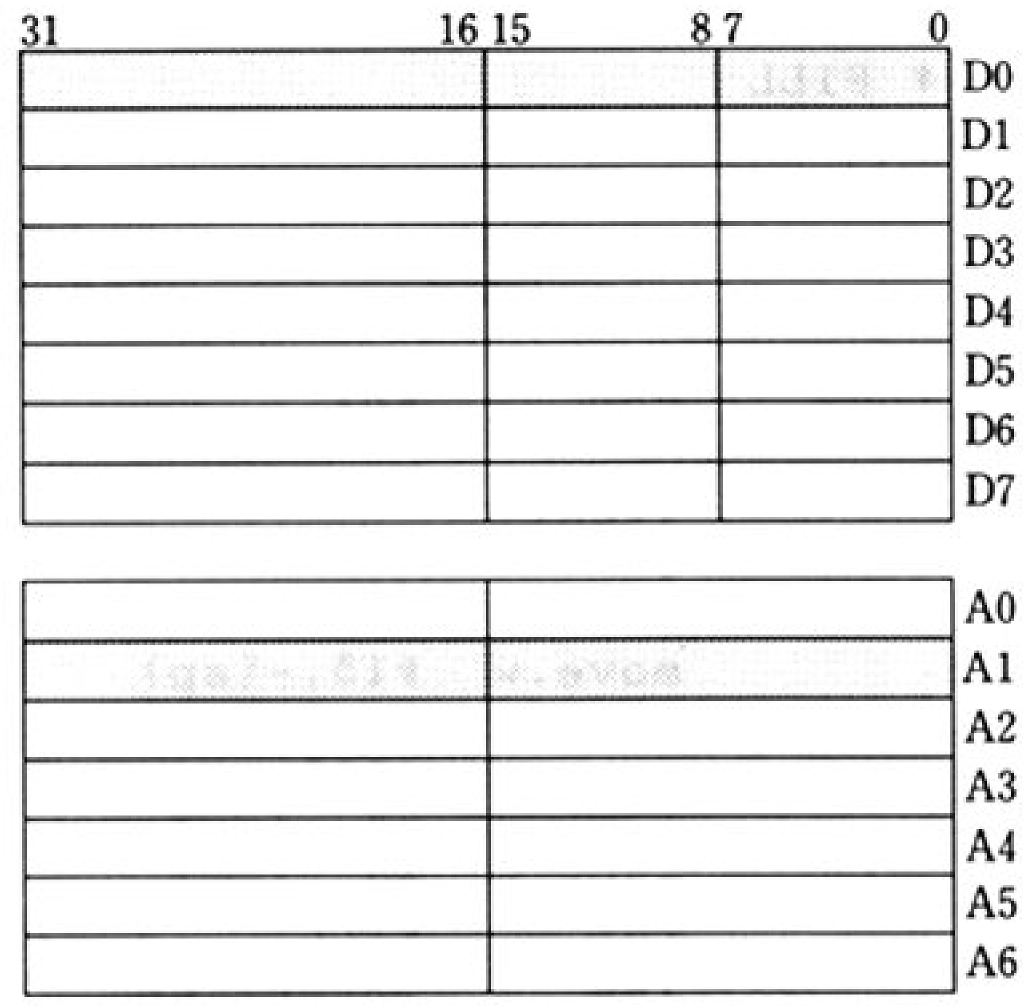
iocs    equ     $0f

        .text
        .
        .                       * fill(0, 0, 767, 511, 15)
        move.w  #15,-(sp)       * パレットコード
        move.w  #511,-(sp)      * Y2
        move.w  #767,-(sp)      * X2
        move.w  #0,-(sp)        * Y1
        move.w  #0,-(sp)        * X1
        bsr     FILL            * サブルーチンを呼びます
        lea     10(sp),sp       * スタックを元に戻します
        .
        .

FILL:                           *+++ Fill +++
        moveq.l #_FILL,d0       * IOCSコール番号をセット
        movea.l sp,a1           *
        addq.l  #4,a1           * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

CIRCLE

● CIRCLE　　　IOCS コール番号：$BB



機　能	グラフィック画面のサークル

入　力		
	D0. L	$BB　IOCS コール番号
	A1. L	パラメータが格納されているアドレスの先頭
	0 (A1). W	中心 X 座標
	2 (A1). W	中心 Y 座標
	4 (A1). W	半径 R
	6 (A1). W	パレットコード
	8 (A1). W	円弧開始角度［°］ 負の値を指定すると扇型を描く
	10 (A1). W	円弧終了角度［°］ 負の値を指定すると扇型を描く
	12 (A1). W	比率 P

0 ≦ P＜256　　：RX ＝ R、RY ＝ R×P／256………横長楕円
256 ＜ P ＜ 65280：RY ＝ R、RX ＝ R×P／65536……縦長楕円
P ＝ 256　　　：RX ＝ RY ＝ R………………………円

リターン　D0. L　　終了状態
　D0. L ＝ 0……正常終了
　D0. L ＝ −1……グラフィック画面使用不可

解　説
グラフィック画面にサークル描画を行います。
入力の A1. L は、パラメータが格納されているアドレスの先頭です。
入力の 0 (A1). W、2 (A1). W、4 (A1). W、6 (A1). W は、それぞれ中心 X 座標、中心 Y 座標、半径 R、パレットコードです。
入力の 8 (A1). W、10 (A1). W は円弧開始、終了角度です。
符号つきワードで、0 から 360 までの値をとり円弧を描きます。
範囲外の値は、360 を指定したこととみなされ、負の値を指定すると扇型（角度は値の絶対値）を描きます。
入力の 12 (A1). W は比率 P です。
符号なしワードで 0 から 65280 までの値をとります。
0 から 255 のとき半径は、横方向の半径 RX を意味し、横長の楕円になります。
縦方向の半径 RY は、半径 R×比率 P／256 になります。

256から65280のとき半径は、縦方向の半径RYを意味し、縦長の楕円になります。

横方向の半径RXは、半径R×比率P／65536になります。

比率Pは、256の倍数になるよう切捨てられます。

円を描くときには、比率Pを256にします。

プログラム例

```
*********************************************
* CIRCLE                                  $BB
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

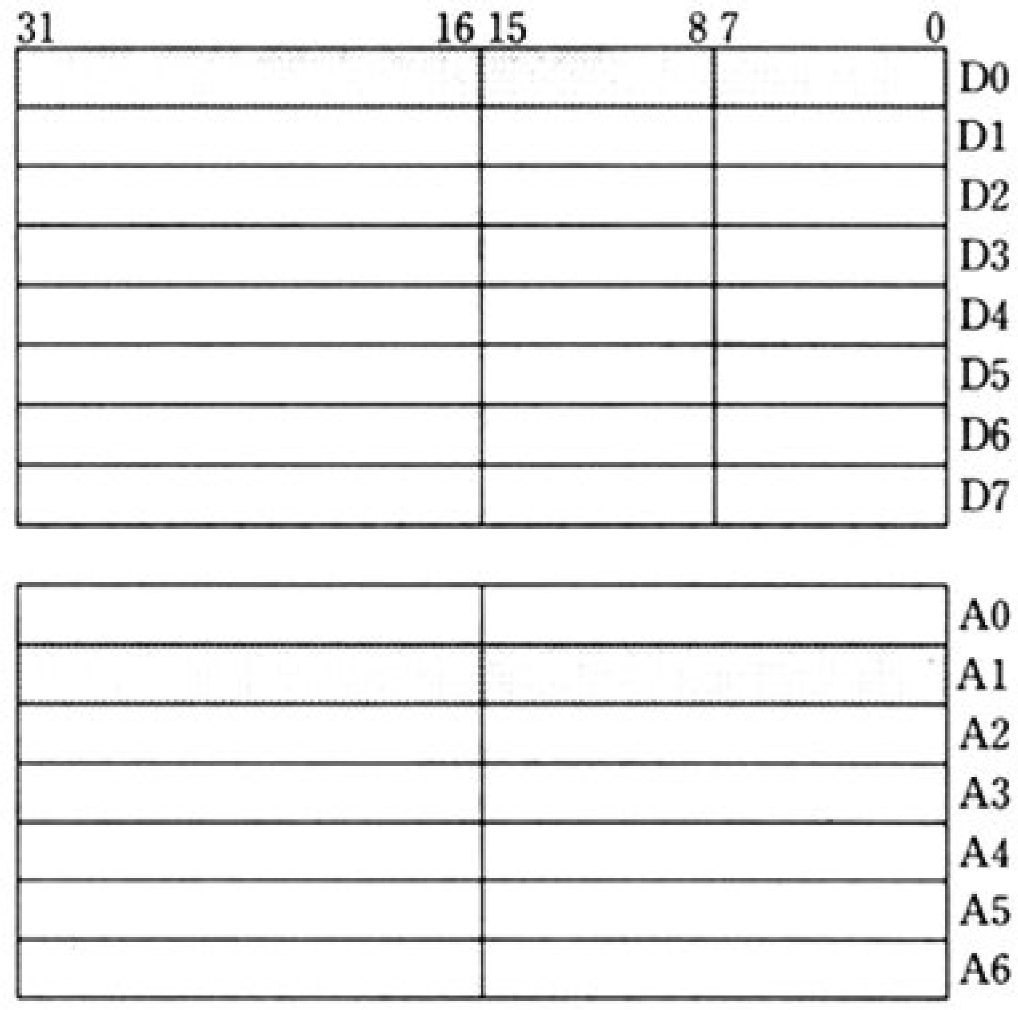
        .text
        .
        .                       * circle(256, 256, 256, 15, 0, 360, 256)
        move.w  #256,-(sp)      * 比率
        move.w  #360,-(sp)      * 終了角度
        move.w  #0,-(sp)        * 開始角度
        move.w  #15,-(sp)       * パレットコード
        move.w  #256,-(sp)      * 半径
        move.w  #256,-(sp)      * 中心 Y
        move.w  #256,-(sp)      * 中心 X
        bsr     CIRCLE          * サブルーチンを呼びます
        lea     14(sp),sp       * スタックを元に戻します
        .
        .

CIRCLE:                         *+++ Circle +++
        moveq.l #_CIRCLE,d0     * IOCSコール番号をセット
        movea.l sp,a1           *
        addq.l  #4,a1           * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

PAINT

● PAINT　　IOCS コール番号：$BC

機　　能　グラフィック画面のペイント



入　　力

D0. L	$BC	IOCS コール番号
A1. L	パラメータが格納されているアドレスの先頭	
0 (A1). W	X 座標	
2 (A1). W	Y 座標	
4 (A1). W	パレットコード	
6 (A1). L	作業領域先頭アドレス	
10 (A1). L	作業領域終了アドレス	

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了

D0. L ＝ −1……グラフィック画面使用不可

解　　説　グラフィック画面のペイント描画を行います。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W、4 (A1). W は、それぞれ X 座標、Y 座標、パレットコードです。

入力の 6 (A1). L、10 (A1). L は、それぞれ作業領域先頭アドレス、終了アドレスです。

プログラム例

```
*******************************************
* PAINT                                 $BC
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

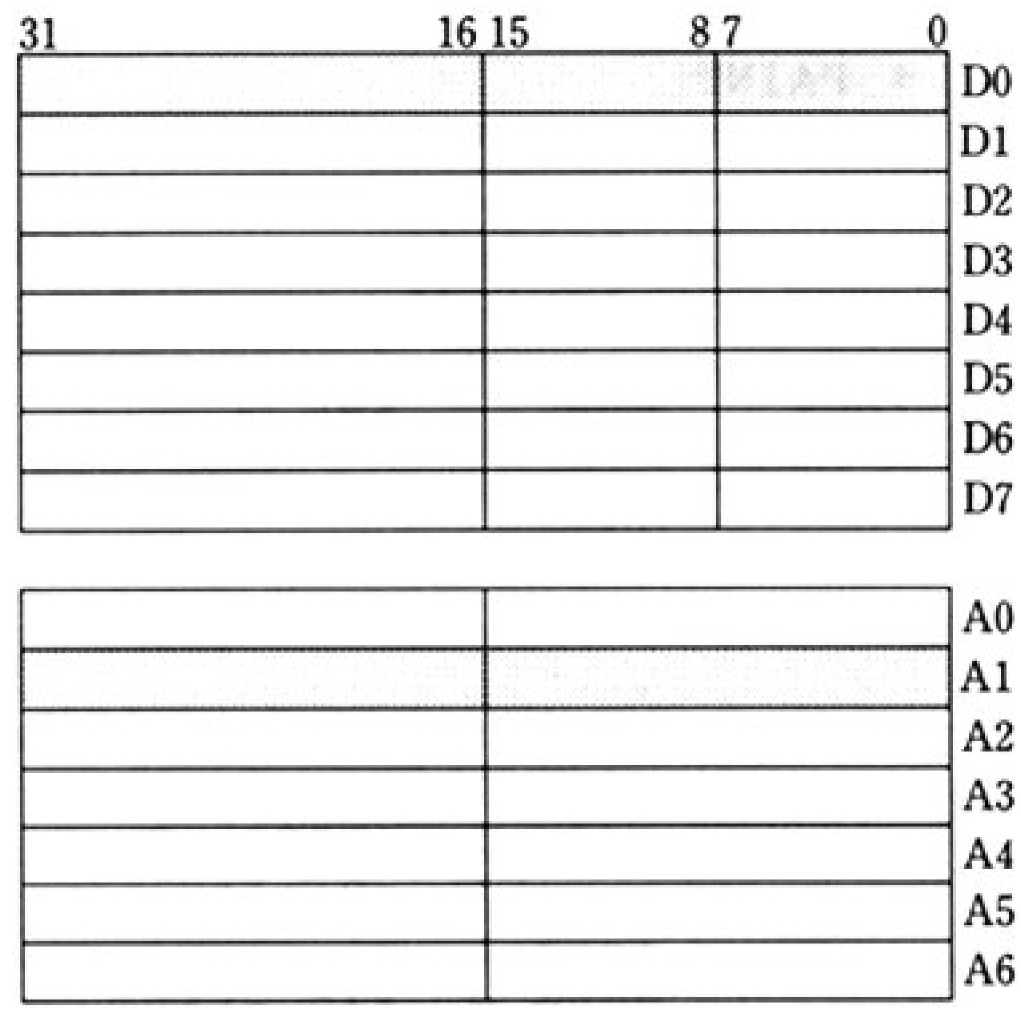
iocs    equ     $0f

        .text
        .
        .
        .                               * paint(100, 200, 15)
        lea     paint_work,a1           * a1 <-- Work Top
        movea.l a1,a2
        adda.l  #1024,a2                * a2 <-- Work Botom
        move.l  a2,-(sp)                * 作業領域終了アドレス
        move.l  a1,-(sp)                * 作業領域先頭アドレス
        move.w  #15,-(sp)               * パレットコード
        move.w  #200,-(sp)              * 中心 Y
        move.w  #100,-(sp)              * 中心 X
        bsr     PAINT                   * サブルーチンを呼びます
        lea     14(sp),sp               * スタックを元に戻します
        .
        .

PAINT:                                  *+++ Paint +++
        moveq.l #_PAINT,d0              * IOCSコール番号をセット
        movea.l sp,a1                   *
        addq.l  #4,a1                   * a1=sp+4 がパラメータテーブルのアドレス
        trap    #iocs                   * IOCSコール
        rts
        .
        .
        .data
        .
        .
paint_work:
        ds.b    1024
        .
        .
        .end
```

SYMBOL

● SYMBOL　　　IOCS コール番号：$BD



機　　能　グラフィック画面のシンボル

入　　力

D0. L	$BD	IOCS コール番号
A1. L	パラメータが格納されているアドレスの先頭	
0 (A1). W	X 座標	
2 (A1). W	Y 座標	
4 (A1). L	文字列先頭アドレス	
8 (A1). B	横倍率	
9 (A1). B	縦倍率	
10 (A1). W	パレットコード	
12 (A1). B	文字フォントのタイプ	
13 (A1). B	角度コード	

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了

D0. L ＝ −1……グラフィック画面使用不可

解　　説　グラフィック画面にシンボル（拡大文字）を表示します。

パターンが画面からはみ出しているかどうかは判定しません。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W は、それぞれ X 座標、Y 座標です。

座標は、通常は表示する文字列の左上隅の座標を指定しますが、回転角度が 90 度、180 度、270 度のときはそれぞれ、左下、右下、右上隅の座標を指定しなければなりません。

入力の 4 (A1). L は、文字列先頭アドレスです。

入力の 8 (A1). B、9 (A1). B はそれぞれ横倍率、縦倍率です。

入力の 10 (A1). W は、パレットコードです。

入力の 12 (A1). B は、文字フォントのタイプです。

内容は次の通りです。

0……全角 12×12

1……全角 16×16

2……全角 24×24

入力の 13(A1). B は角度コードです。

内容は次の通りです。

0……通常
1……90 度回転
2……180 度回転
3……270 度回転

文字列は、シフト JIS 漢字文字列です。

プログラム例

```
*******************************************
* SYMBOL                                $BD
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

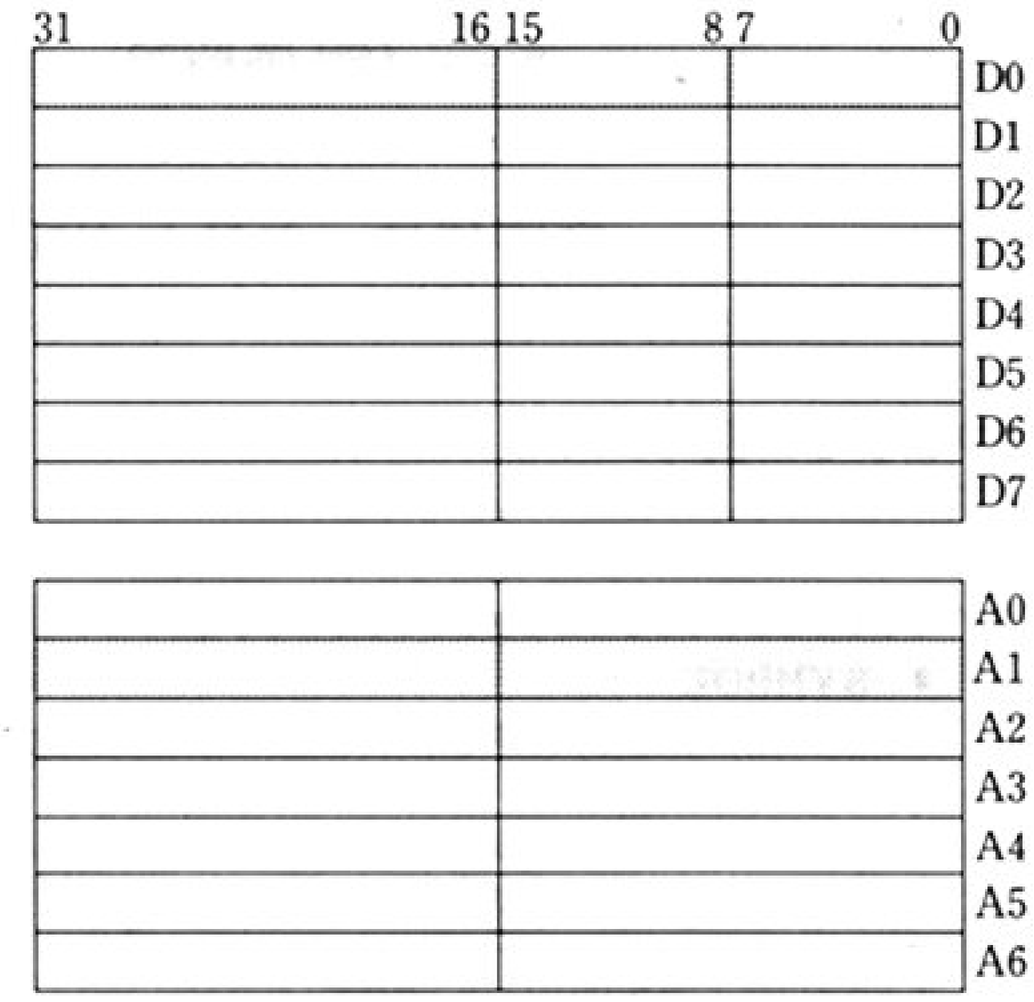
        .text
        .
        .
        moveq.l #_SYMBOL,d0     * IOCSコール番号をセット
        lea     work_symbol,a1
        lea     str_symbol,a2
        move.b  #0,13(a1)       * 角度コード
        move.b  #2,12(a1)       * フォントタイプ
        move.w  #15,10(a1)      * パレットコード
        move.b  #1,9(a1)        * 縦倍率
        move.b  #1,8(a1)        * 横倍率
        move.l  a2,4(a1)        * 文字列先頭アドレス
        move.w  #0,2(a1)        * 左上隅の Y 座標
        move.w  #0,0(a1)        * 左上隅の X 座標
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .
        .

work_symbol:
                ds.b    14
str_symbol:
                dc.b    "漢字ですよαβγ",0
        .
        .
        .end
```

GETGRM

● GETGRM　　IOCS コール番号：$BE

機　能　グラフィック画面読み込み



入　力

D0. L	$BE	IOCS コール番号
A1. L	パラメータが格納されているアドレスの先頭	
0 (A1). W	始点 X 座標	
2 (A1). W	始点 Y 座標	
4 (A1). W	終点 X 座標	
6 (A1). W	終点 Y 座標	
8 (A1). L	バッファ先頭アドレス	
12 (A1). L	バッファ終了アドレス	

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了

D0. L ＝ −1……グラフィック画面使用不可

解　説　グラフィック画面から読み込みます。

入力の A1. L は、パラメータが格納されているアドレスの先頭です。

入力の 0 (A1). W、2 (A1). W はそれぞれ始点の X 座標、Y 座標です。

入力の 4 (A1). W、6 (A1). W はそれぞれ終点の X 座標、Y 座標です。

入力の 8 (A1). L、12 (A1). L はそれぞれバッファ先頭アドレス、バッファ終了アドレスです。

バッファにはパックされた形で、データそのものが入ります。

プログラム例

```
******************************************
* GETGRM                               $BE
******************************************

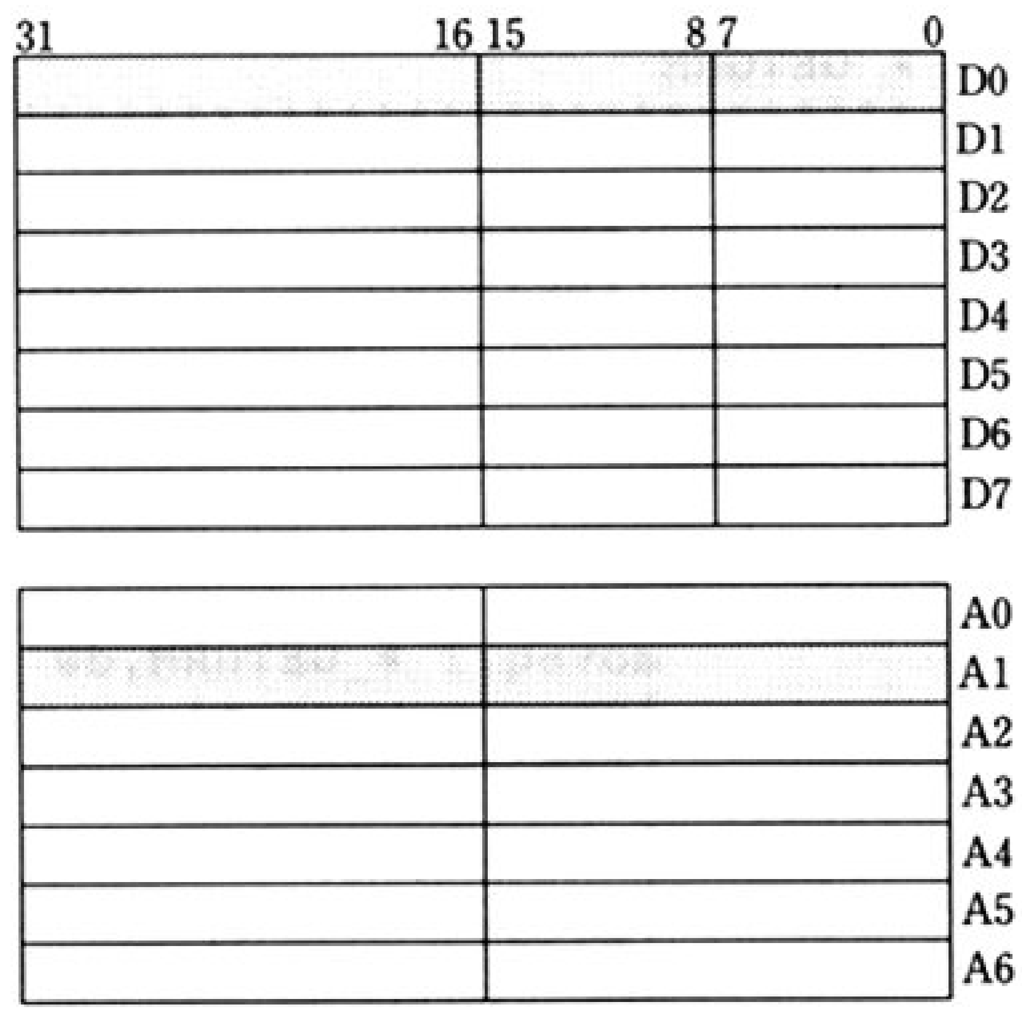
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_GETGRM,d0     * IOCSコール番号をセット
        lea     work,a1
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .
        .
work:
        dc.w    0,0,64,64
        dc.l    buftop,bufend
buftop:
        ds.w    4096
bufend:
        .
        .
        .end
```

PUTGRM

● PUTGRM　　IOCS コール番号：$BF



機　　能　グラフィック画面書き込み

入　　力

D0. L　　$BF　　IOCS コール番号
A1. L　　パラメータが格納されているアドレスの先頭
0 (A1). W　　始点 X 座標
2 (A1). W　　始点 Y 座標
4 (A1). W　　終点 X 座標
6 (A1). W　　終点 Y 座標
8 (A1). L　　バッファ先頭アドレス
12 (A1). L　　バッファ終了アドレス

リターン

D0. L　　終了状態
　D0. L ＝ 0………正常終了
　D0. L ＝ −1……グラフィック画面使用不可

解　　説

グラフィック画面へ書き込みます。
入力の A1. L は、パラメータが格納されているアドレスの先頭です。
入力の 0 (A1). W、2 (A1). W はそれぞれ始点の X 座標、Y 座標です。
入力の 4 (A1). W、6 (A1). W はそれぞれ終点の X 座標、Y 座標です。
入力の 8 (A1). L、12 (A1). L はそれぞれバッファ先頭アドレス、バッファ終了アドレスです。
バッファにはパックされた形で、データそのものを入れておいてください。

プログラム例

```
********************************************
* PUTGRM                                $BF
********************************************

        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

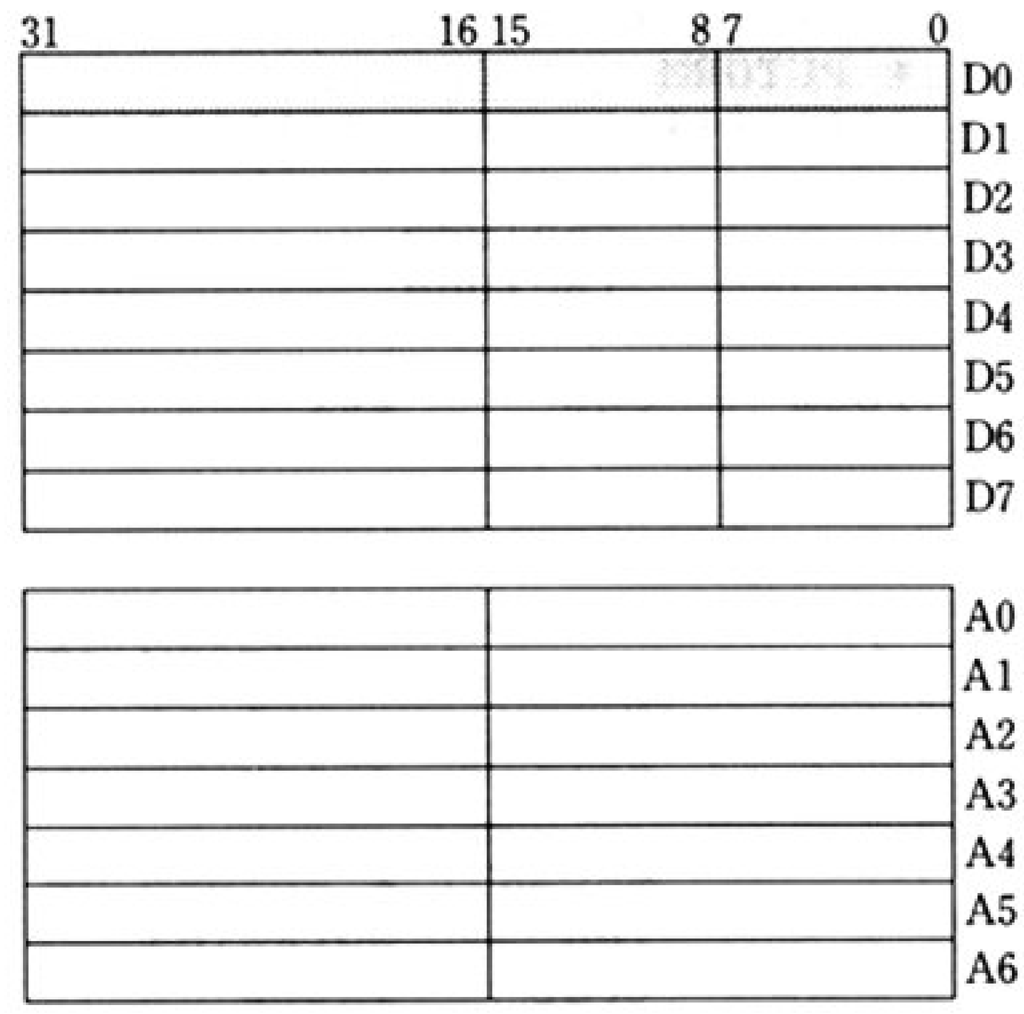
        .text
        .
        .
        moveq.l #_GETGRM,d0      * IOCSコール番号をセット
        lea     work1,a1
        trap    #iocs            * IOCSコール

        moveq.l #_PUTGRM,d0      * IOCSコール番号をセット
        lea     work2,a1
        trap    #iocs            * IOCSコール
        .
        .
        .data
        .
        .
work1:
        dc.w    0,0,64,64
        dc.l    buftop,bufend
work2:
        dc.w    65,65,128,128
        dc.l    buftop,bufend
buftop:
        ds.w    4096
bufend:
        .
        .
        .end
```

SP_INIT

● SP_INIT　　IOCS コール番号：$C0

機　　能　スプライト面の初期化



入　　力　D0. L　　$C0　　IOCS コール番号

リターン　D0. L　　終了状態

D0. L ＝ 0……正常終了

D0. L ＝ −1……画面モードが不正

解　　説　スプライト面を初期化します。

プログラム例

```
*******************************************
* SP_INIT                              $C0
*******************************************

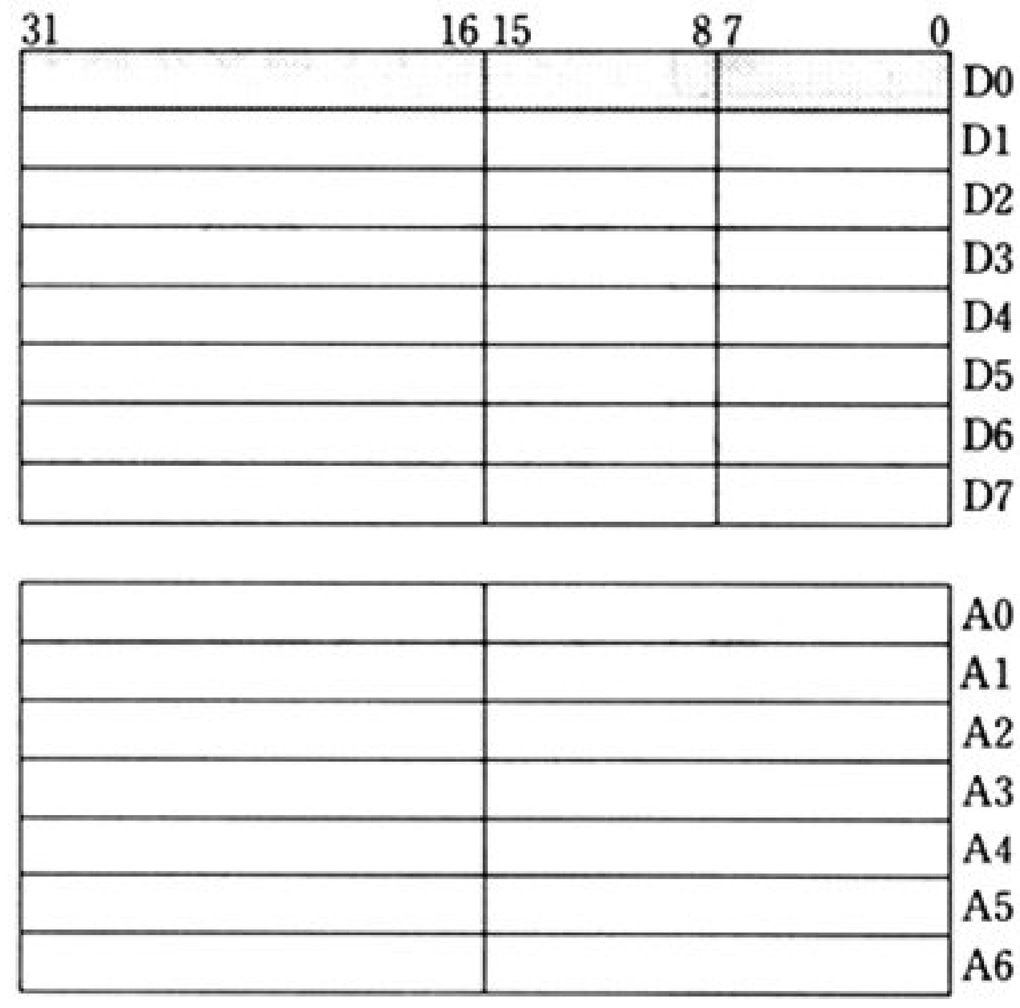
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_INIT,d0    * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

SP_ON

● SP_ON　　IOCS コール番号：$C1



機　能　スプライト面の表示

入　力　D0. L　$C1　IOCS コール番号

リターン　D0. L　終了状態

D0. L ＝ 0　……正常終了

D0. L ＝ −1　…画面モードが不正

解　説　スプライト面を表示します。

プログラム例

```
********************************************
* SP_ON                                 $C1
********************************************

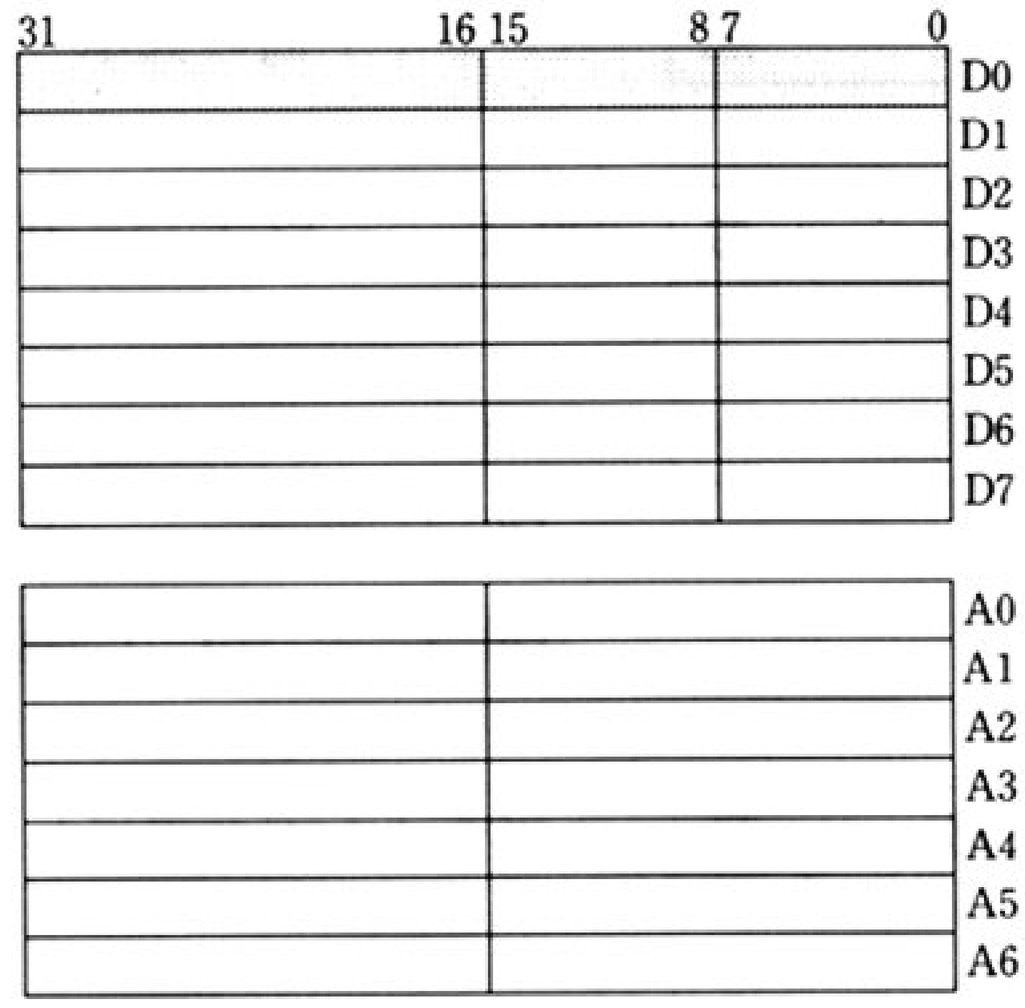
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_ON,d0      * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

SP_OFF

● SP_OFF　　IOCS コール番号：$C2



機　　能　スプライト面の非表示

入　　力　D0. L　　$C2　　IOCS コール番号

リターン　なし（D0. L の値は保証されません）

解　　説　スプライト面を表示しません。

プログラム例

```
********************************************
* SP_OFF                               $C2
********************************************

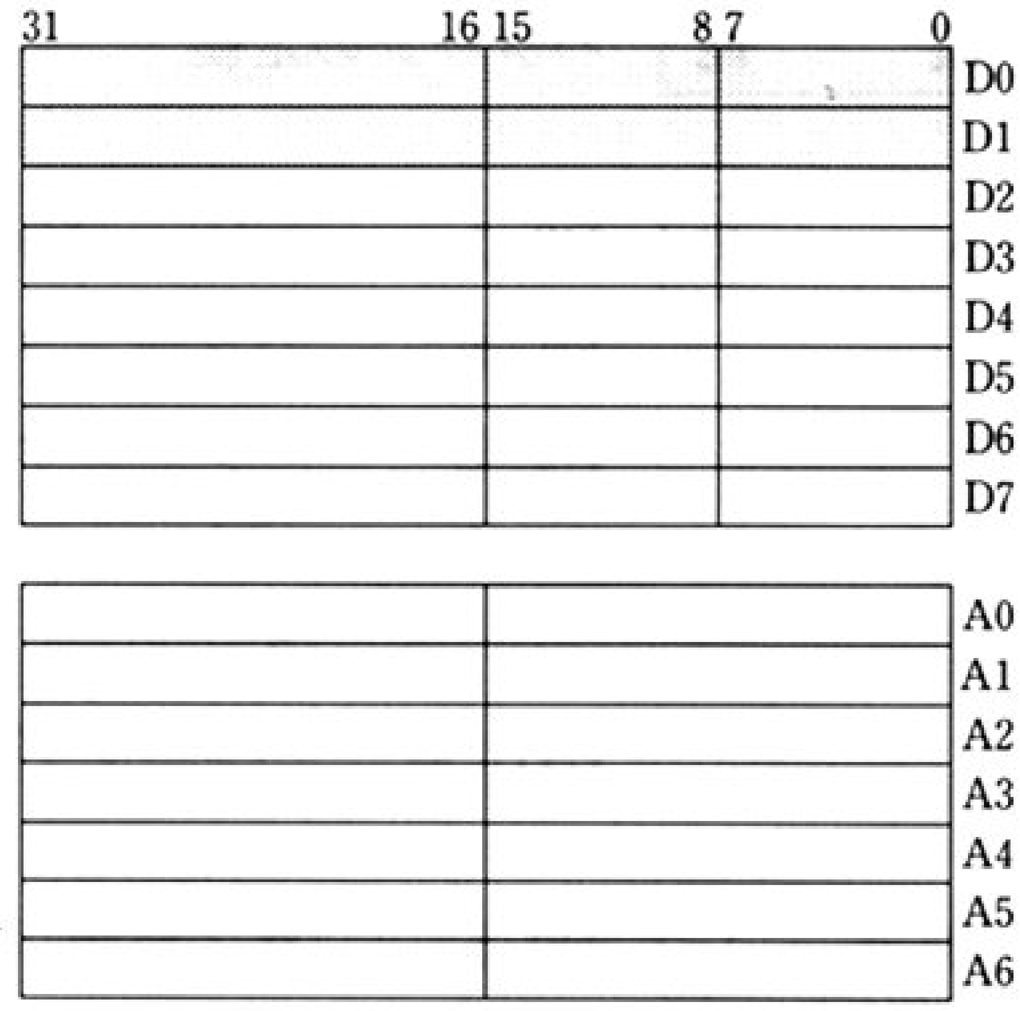
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_OFF,d0     * IOCSコール番号をセット
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

SP_CGCLR

● SP_CGCLR　　　IOCS コール番号：$C3



機　能　パターンのクリア

入　力　D0. L　　$C3　IOCS コール番号
D1. L　　パターンコード

リターン　D0. L　　終了状態
D0. L ＝ 0………正常終了
D0. L ＝ −1……画面モードが不正

解　説　スプライトやバックグラウンドのパターンをクリアします。
入力の D1. L はパターンコードです。
0〜255 の値をとります。

プログラム例

```
********************************************
* SP_CGCLR                             $C3
********************************************

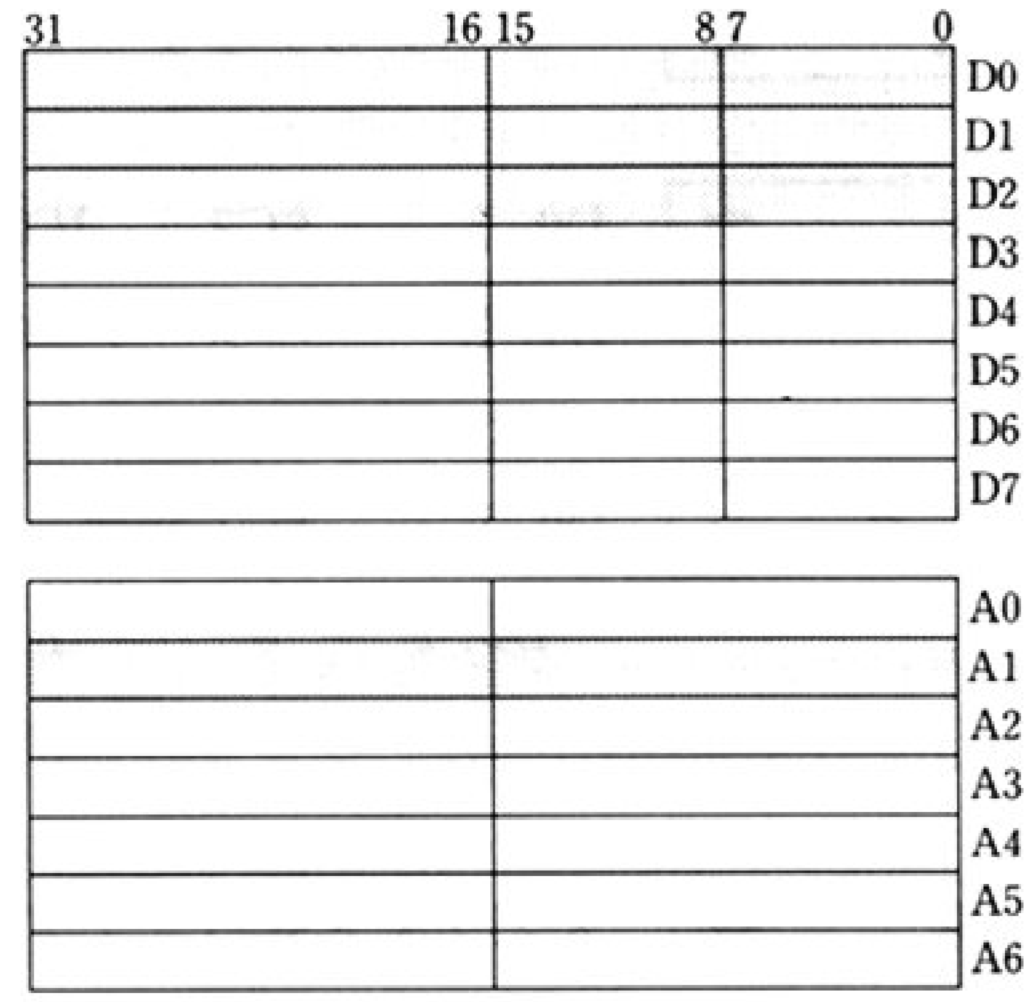
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_CGCLR,d0   * IOCSコール番号をセット
        moveq.l #64,d1          * パターンコード 64 番をクリア
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

SP_DEFCG

● SP_DEFCG　　　IOCS コール番号：$C4



機　　能　パターンの定義

入　　力

D0. L　　$C4　IOCS コール番号
D1. L　　パターンコード
D2. L　　パターンサイズ
A1. L　　パターンデータの先頭アドレス
(A1). B　パターンデータ

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了
D0. L ＝ −1……画面モードが不正

解　　説

スプライトやバックグラウンドのパターンを定義します。
入力の D1. L は、パターンコードです。
0〜255 の値をとります。
入力の D2. L は、パターンサイズです。
内容は次の通りです。

D2. L ＝ 0……8×8 ドットのパターン
D2. L ＝ 1……16×16 ドットのパターン

入力の A1. L は、パターンデータの先頭アドレスです。
入力の(A1). B は、パターンデータです。
D2. L が 0 のときは 32 バイト、D2. L が 1 のときは 128 バイトのパターンデータを指定します。

プログラム例

```
*********************************************
* SP_DEFCG                               $C4
*********************************************

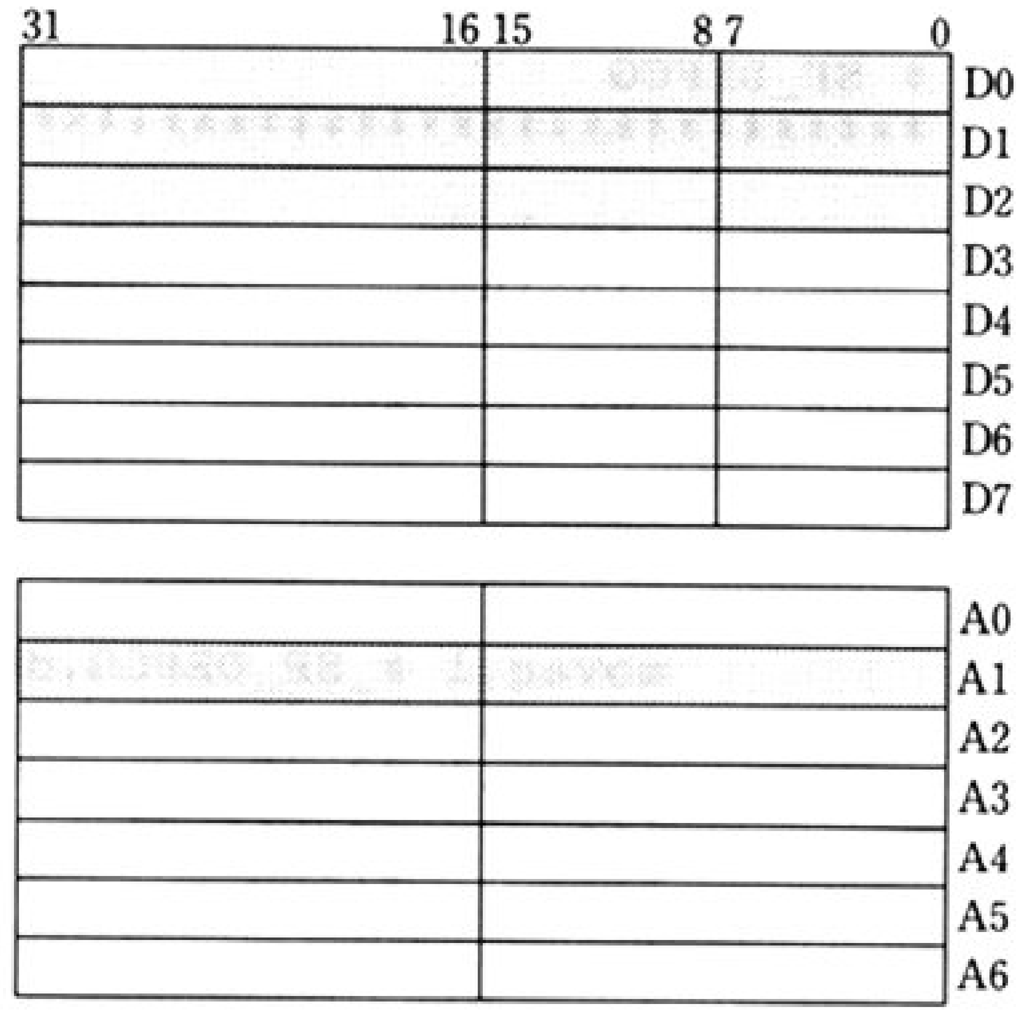
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_DEFCG,d0   * IOCSコール番号をセット
        moveq.l #1,d1           * パターンコード 1 番を定義
        moveq.l #0,d2           * 8×8 ドットのパターン
        lea     sp_pat,a1       * パターンデータ先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .
        .
sp_pat: dc.l    $00000000       * 1 ドットごとに 16 色指定できます
        dc.l    $11200011       * 1 バイトが 2 ドットに相当するので
        dc.l    $00120122       * 8×8 のパターンだと
        dc.l    $00011200       * 8 * 8 / 2 = 32Byte = 8 Long Word
        dc.l    $00011200
        dc.l    $00122120
        dc.l    $11200011
        dc.l    $22200022
        .
        .
        .end
```

SP_GTPCG

● SP_GTPCG　　IOCS コール番号：$C5



機　　能　パターンの読み込み

入　　力

D0. L　　$C5　IOCS コール番号
D1. L　　パターンコード
D2. L　　パターンサイズ
A1. L　　パターンデータの先頭アドレス

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了
D0. L ＝ −1……画面モードが不正

(A1). B　読み込まれたパターンデータ

解　　説

スプライトやバックグラウンドのパターンを読み出します。
入力の D1. L は、パターンコードです。
0〜255 の値をとります。
入力の D2. L は、パターンサイズです。
内容は次の通りです。

D2. L ＝ 0……8×8 ドットのパターン
D2. L ＝ 1……16×16 ドットのパターン

入力の A1. L は、パターンデータの先頭アドレスです。
リターンの(A1). B は、読み込まれたパターンデータです。
D2. L が 0 のときは 32 バイト、D2. L が 1 のときは 128 バイトのパターンデータ用バッファを確保する必要があります。

プログラム例

```
*******************************************
* SP_GTPCG                            $C5
*******************************************

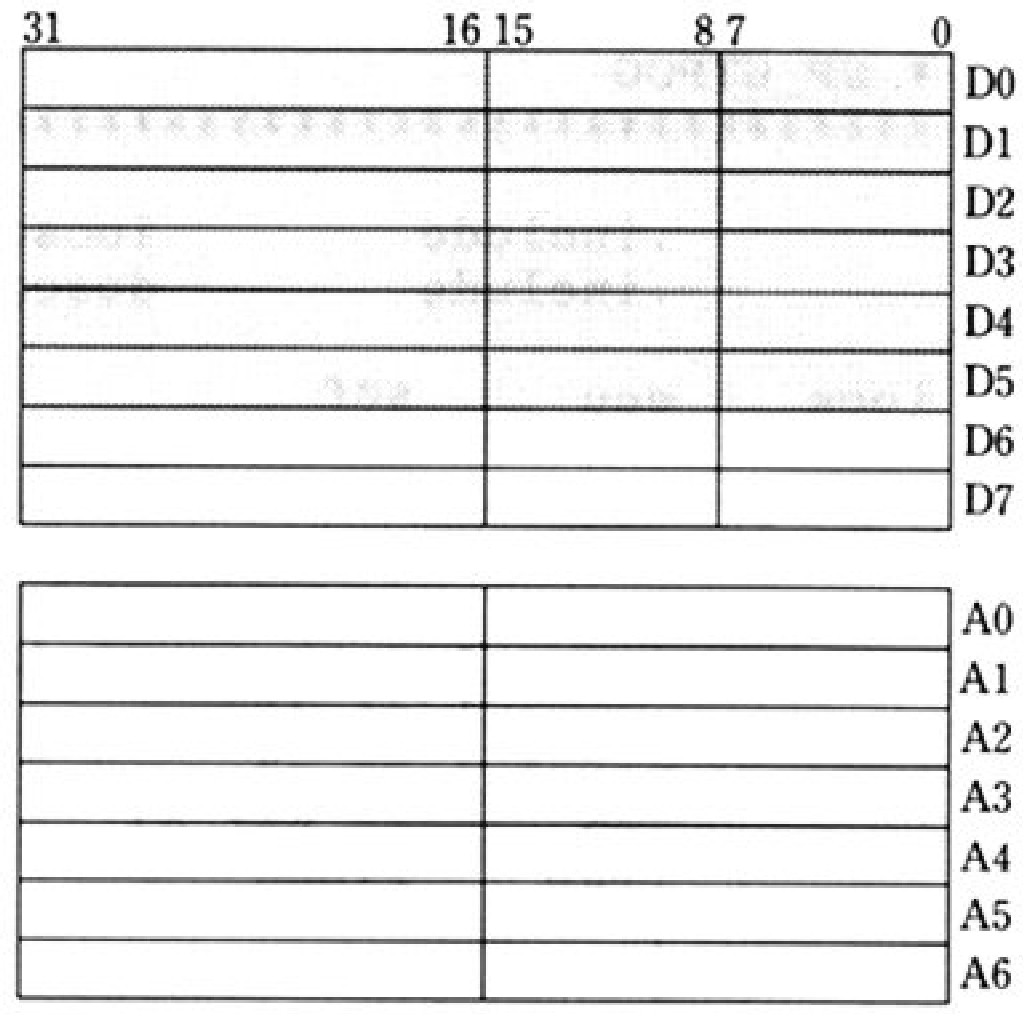
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_GTPCG,d0   * IOCSコール番号をセット
        moveq.l #1,d1           * パターンコード1番のデータを読む
        moveq.l #0,d2           * 8×8 ドットのパターン
        lea     sp_pat,a1       * データバッファ先頭アドレス
        trap    #iocs           * IOCSコール
        .
        .
        .data
        .
        .
sp_pat: ds.l    8
        .
        .
        .end
```

SP_REGST

● SP_REGST　　　IOCS コール番号：$C6



機　　能　スプライトレジスタの設定

入　　力

D0. L　　$C6　　IOCS コール番号

D1. L　　スプライト番号、垂直帰線期間の検出の有無

D2. L　　X 座標

D3. L　　Y 座標

D4. L　　パターンコード

D5. L　　プライオリティ

リターン　D0. L　　終了状態

D0. L ＝ 0………正常終了

D0. L ＝ −1……画面モードが不正

解　　説　スプライトレジスタを設定します。

入力の D1. L の内容を次に示します。

ビット 31：垂直帰線期間の検出の有無

0……検出後設定

1……検出なし

ビット 6~0：スプライト番号

値は 0～127 です。

入力の D2. L、D3. L はそれぞれ X 座標、Y 座標です。

値は 0～1023 ですが、−1 を指定すると前回指定した値を利用します。

ただし、初期指定のときに−1 を指定してはいけません。

入力の D4. L はパターンコードです。

パターンコードは、次の 4 つのデータを各ビットに割りあてロングワード 1 つで指定します。

−1 を指定すると前回指定した値を利用しますが、初期指定のときに−1 を指定してはいけません。

−1 以外の内容は次の通りです。

ビット 31～16：常に0

ビット 15：　　V（縦方向）

0……反転しない
1……反転する

ビット14：　H（横方向）
0……反転しない
1……反転する

ビット13～12：常に0

ビット11～8：COLOR（パレットブロック0～15）
1つのパレットに16種類の色が指定できます。
パレットブロックは16個あり、そのパレットブロックの番号を指定します。

ビット7～0：CODE（パターンコード0～255）

入力のD5.Lは、プライオリティです。
−1を指定すると前回指定した値を利用しますが、初期指定のときに−1を指定してはいけません。
−1以外の内容は次の通りです。

0……スプライトを表示しません
1……スプライトがBG0、BG1より後
2……スプライトがBG0とBG1の間
3……スプライトがBG0、BG1より前

プログラム例

```
********************************************
* SP_REGST                              $C6
********************************************

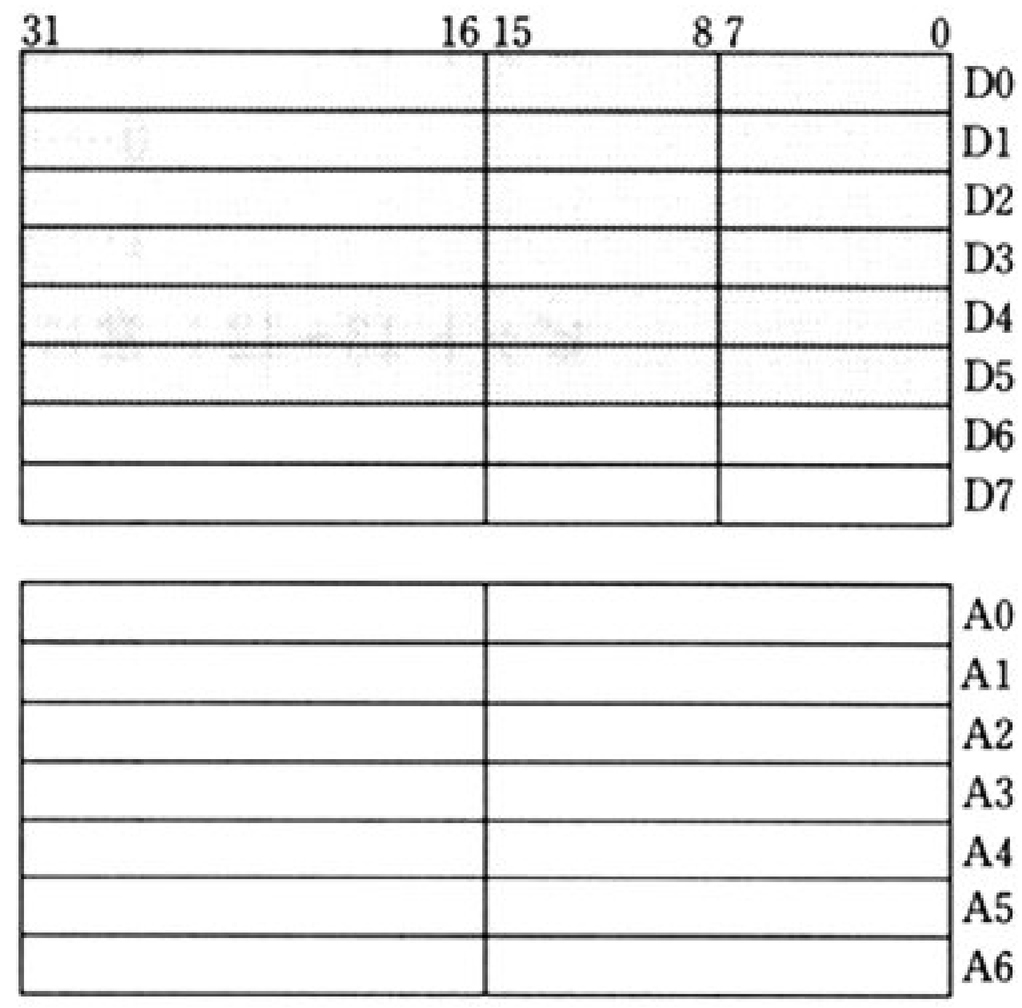
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_REGST,d0    * IOCSコール番号をセット
        move.l  #$8000_0001,d1   * スプライト 1 番を使用
        move.l  #200,d2          * X 200
        moveq.l #100,d3          * Y 100
        move.l  #$01_01,4        * パレットブロック 1、パターンコード 1
        moveq.l #3,d5            * プライオリティ 3
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

SP_REGGT

● SP_REGGT　　　IOCS コール番号：$C7



機　能　スプライトレジスタ読み出し

入　力　D0. L　　$C7　IOCS コール番号
D1. L　　スプライト番号

リターン　D0. L　　終了ステータス
　D0. L ＝ 0………正常終了
　D0. L ＝ −1……画面モードが不正
D2. L　　X 座標 (0〜1023)
D3. L　　Y 座標 (0〜1023)
D4. L　　パターンコード
D5. L　　プライオリティ

解　説　スプライトレジスタを読み出します。
入力の D1. L はスプライト番号です。
0〜127 の値をとります。
リターンの D4. L のパターンコードは、次の 4 つのデータを各ビットに割りあてられたロングワードです。
内容は次の通りです。

ビット 31〜16：常に 0
ビット 15：　V (縦方向)
　0……反転していない
　1……反転している
ビット 14：　(横方向)
　0……反転していない
　1……反転している
ビット 13〜12：常に 0
ビット 11〜 8：COLOR (パレットブロック 0〜15)
ビット 7〜 0：CODE (パターンコード 0〜255)

リターンの D5. L はプライオリティです。
　D5. L ＝ 0……スプライトを表示しない
　D5. L ＝ 1……スプライトが BG0、BG1 より後
　D5. L ＝ 2……スプライトが BG0 と BG1 の間

D5. L ＝ 3……スプライトがBG0、BG1より前

プログラム例

```
*********************************************
* SP_REGGT                               $C7
*********************************************

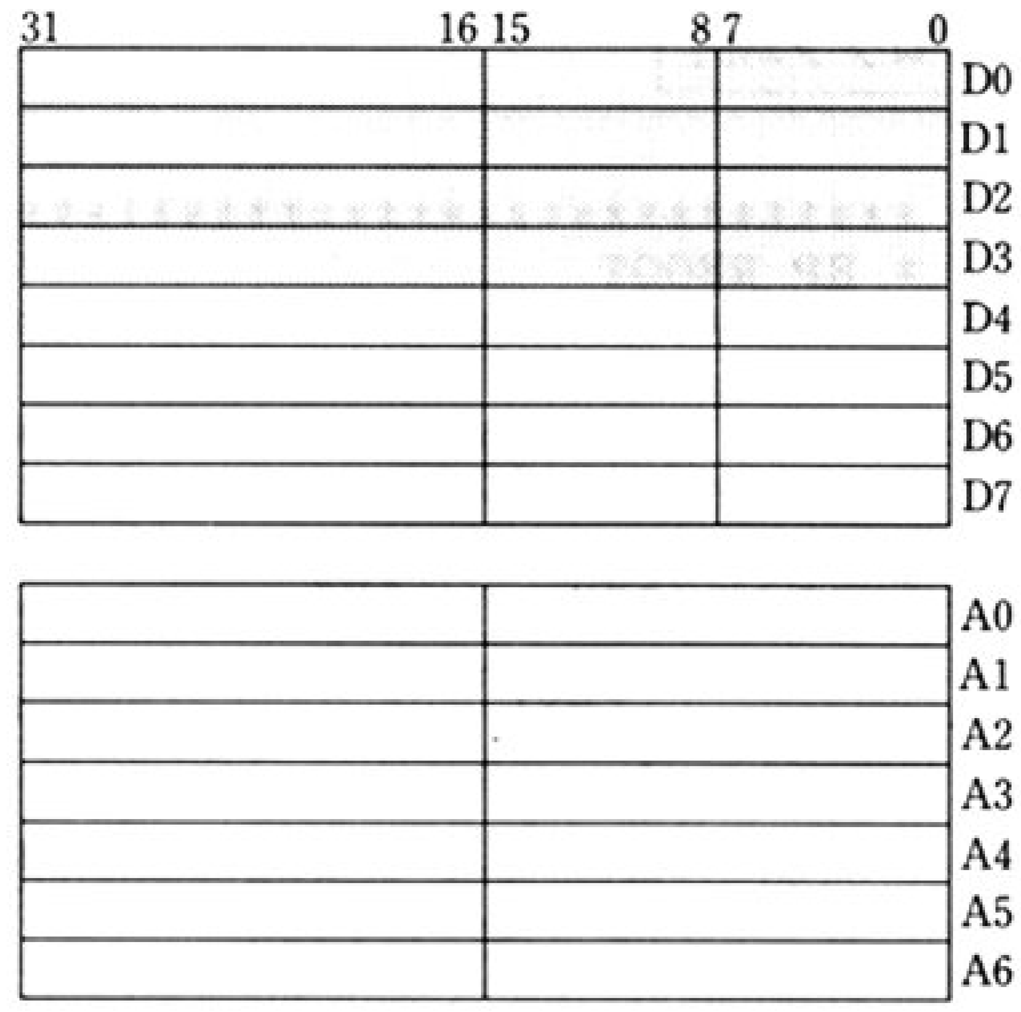
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_SP_REGET,d0    * IOCSコール番号をセット
        moveq.l #1,d1            * スプライト 1 番を調査
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

BGSCRLST

● BGSCRLST　　IOCS コール番号：$C8



機　　能　バックグラウンドスクロールレジスタの設定

入　　力
D0. L　　$C8　　IOCS コール番号
D1. L　　バックグラウンド
D2. L　　X 座標
D3. L　　Y 座標

リターン
D0. L　　終了状態
　D0. L ＝ 0………正常状態
　D0. L ＝ −1……画面モードが不正

解　　説　バックグラウンドスクロールレジスタを設定します。
入力の D1. L の内容を次に示します。

ビット 31：垂直帰線期間の検出の有無
　　0……検出後設定する
　　1……検出せずにすぐ設定する
ビット 0：バックグラウンド
　　0……バックグラウンド 0 に設定
　　1……バックグラウンド 1 に設定

入力の D2. L、D3. L はそれぞれ X 座標、Y 座標です。
0～1023 の範囲内で指定します。
−1 を指定すると前回指定した値を利用しますが、初期指定のときに−1 を指定してはいけません。

プログラム例

```
*******************************************
* BGSCRLST                             $C8
*******************************************

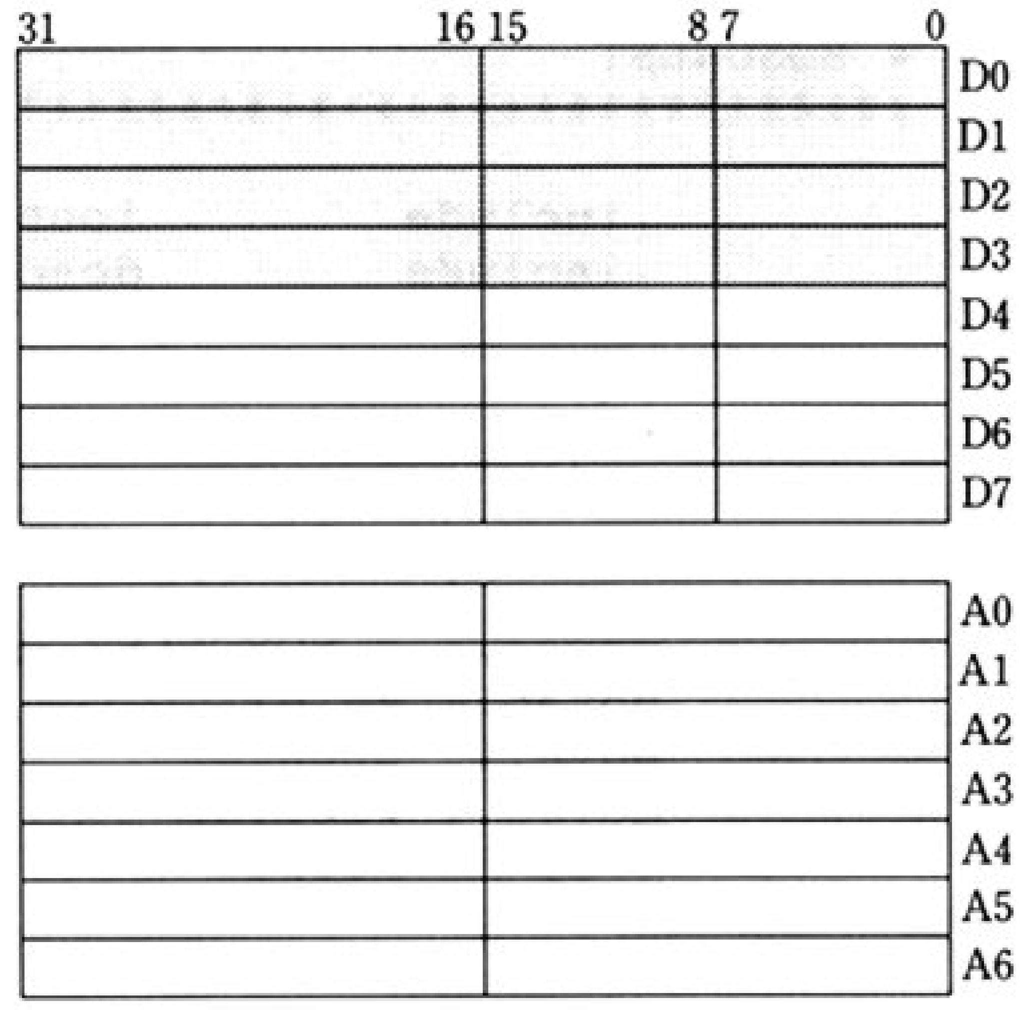
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
        moveq.l #_BGSCRLST,d0    * IOCSコール番号をセット
        moveq.l #0,d1            * バックグラウンド 0
        move.l  #200,d2          * X 200
        moveq.l #100,d3          * Y 100
        trap    #iocs            * IOCSコール
        .
        .
        .end
```

BGSCRLGT

● BGSCRLGT　　　IOCS コール番号：$C9



機　能　バックグラウンドスクロールレジスタの読み出し

入　力
D0. L　　$C9　　IOCS コール番号
D1. L　　バックグラウンド

リターン
D0. L　　終了状態
　D0. L ＝ 0………正常終了
　D0. L ＝ −1……画面モードが不正
正常終了の場合
D2. L　　X 座標 (0～1023)
D3. L　　Y 座標 (0～1023)

解　説　バックグラウンドのスクロールレジスタを読み出します。
入力の D1. L は 0 ならばバックグラウンド 0 を、1 ならばバックグラウンド 1 を読み出します。

プログラム例

```
*********************************************
* BGSCRLGT                              $C9
*********************************************

        .include        iocscall.mac
        .include        doscall.mac

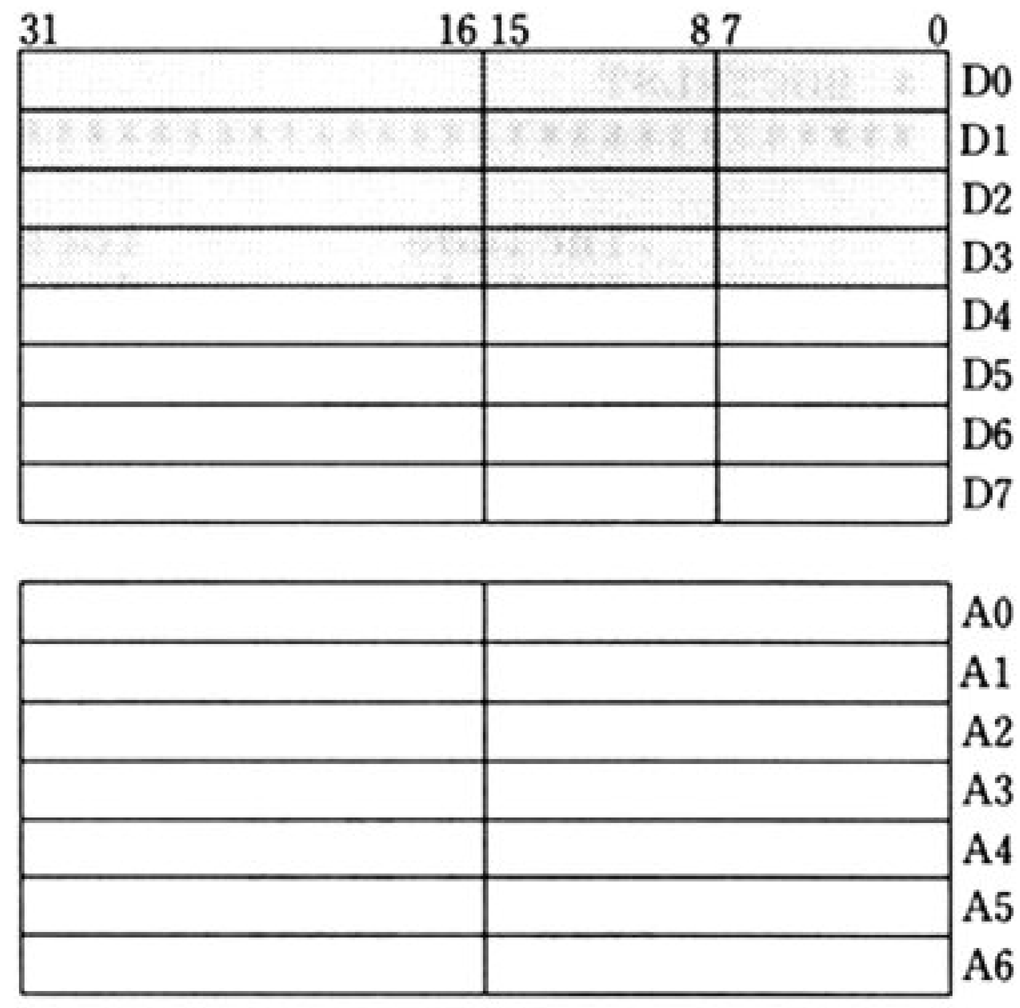
iocs    equ     $0f

        .text
        .
        .
        moveq.l #_BGSCRLST,d0   * IOCSコール番号をセット
        moveq.l #0,d1           * バックグラウンド 0 指定
        trap    #iocs           * IOCSコール
        .
        .
        .end
```

BGCTRLST

● BGCTRLST　　　IOCSコール番号：$CA

機　　能　バックグラウンドコントロールレジスタの設定



入　　力
D0. L　　$CA　　IOCSコール番号
D1. L　　バックグラウンド
D2. L　　テキストページ指定
D3. L　　表示指定

リターン　D0. L　　終了状態
D0. L ＝ 0………正常画面
D0. L ＝ −1……画面モードが不正

解　　説　バックグラウンドのコントロールレジスタを設定します。
入力のD1. Lは0ならバックグラウンド0を、1ならバックグラウンド1を指定します。
入力のD2. Lは0ならテキストページ0を、1ならテキストページ1を指定します。
また−1を指定すると前回指定した値を利用しますが、初期指定のときに−1を指定してはいけません。
入力のD3. Lは0なら非表示を、1なら表示を指定します。
また−1を指定すると前回指定した値を利用しますが、初期指定のときに−1を指定してはいけません。

BGCTRLST

プログラム例

```
*******************************************
* BGCTRLST                             $CA
*******************************************

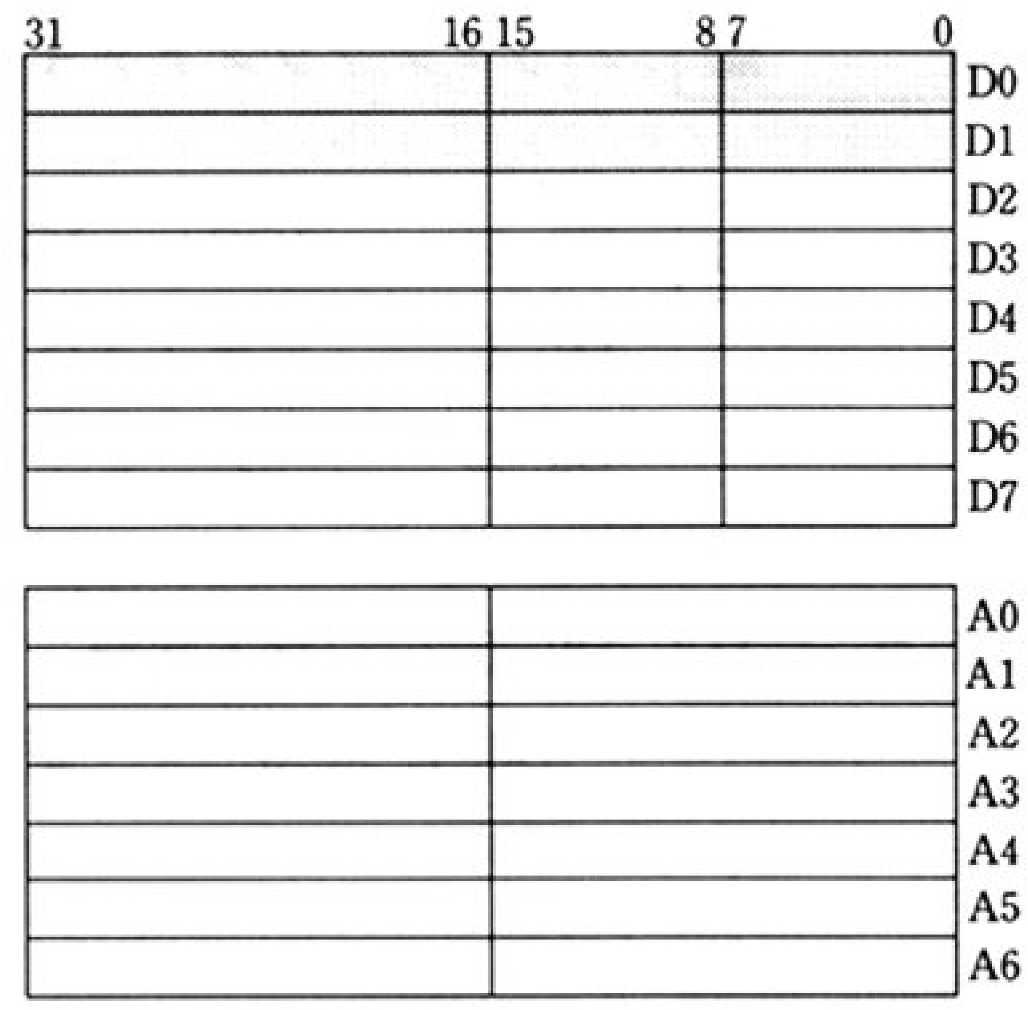
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
BGCTRLST:
        moveq.l #_BGCTRLST,d0    * IOCSコール番号をセット
        moveq.l #0,d1            * バックグラウンド 0 指定
        moveq.l #0,d2            * テキストページ 0
        moveq.l #1,d3            * 表示
        trap    #iocs            * IOCSコール
        rts
        .
        .
        .end
```

BGCTRLGT

● BGCTRLGT　　IOCS コール番号：$CB



機　能　バックグラウンドコントロールレジスタの読み出し

入　力
D0. L　　$CB　　IOCS コール番号
D1. L　　バックグラウンド

リターン
D0. L　　テキストページ
D0. L ＝ 0…テキストページ 0 で非表示
D0. L ＝ 1…テキストページ 0 で表示
D0. L ＝ 2…テキストページ 1 で非表示
D0. L ＝ 3…テキストページ 1 で表示
D0. L ＝ −1…画面モードが不正

解　説
バックグラウンドのコントロールレジスタを読み出します。
入力の D1. L が、0 ならバックグラウンド 0 を、1 ならバックグラウンド 1 を読み出します。

プログラム例

```
********************************************
* BGCTRLGT                             $CB
********************************************

        .include        iocscall.mac
        .include        doscall.mac

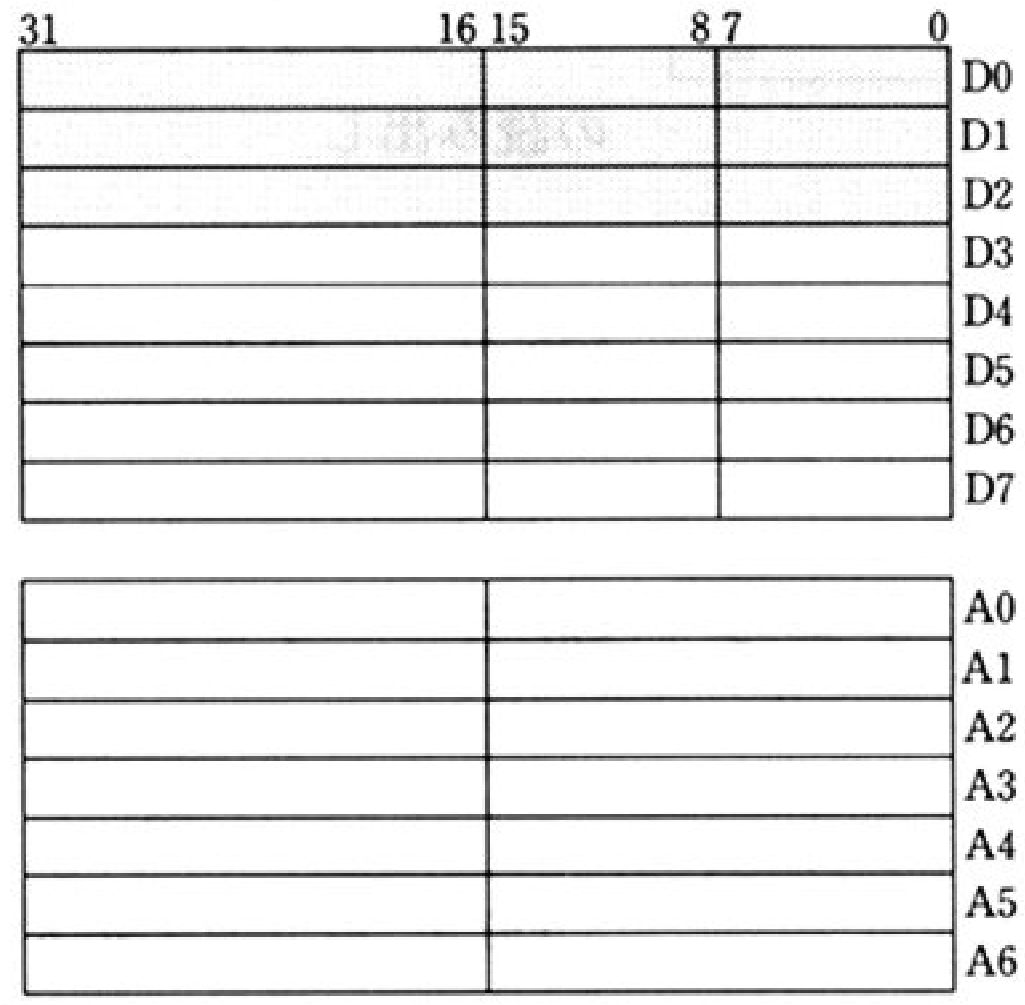
iocs    equ     $0f

        .text
        .
        .
BGCTRLGT:
        moveq.l #_BGCTRLGT,d0    * IOCSコール番号をセット
        moveq.l #0,d1            * バックグラウンド 0 指定
        trap    #iocs            * IOCSコール
        rts
        .
        .
        .end
```

BGTEXTCL

● BGTEXTCL　　IOCS コール番号：$CC

機　　能　バックグラウンドテキストのクリア



入　　力　D0. L　　$CC　　IOCS コール番号
D1. L　　テキストページ
D2. L　　パターンコード

リターン　D0. L　　終了状態
D0. L ＝ 0………正常終了
D0. L ＝ −1……画面モードが不正

解　　説　バックグラウンドのテキストをクリアします。
指定したテキストページを、パターンコードで埋めます。
入力の D1. L は 0 ならばテキストページ 0 を、1 ならばテキストページ 1 を指定します。
入力の D2. L はパターンコードです。
パターンコードは、次の 4 つのデータを各ビットに割りあてられたロングワード 1 つです。
内容は次の通りです。

ビット 31〜16：常に 0
ビット 15：　V（縦方向）
　0……反転しない
　1……反転する
ビット 14：　H（横方向）
　0……反転しない
　1……反転する
ビット 13〜12：常に 0
ビット 11〜 8：COLOR（パレットブロック 0〜15）
　1 つのパレットに 16 種類の色が指定できます。
　パレットブロックは 16 個あり、そのパレットブロックの番号を指定します。
ビット 7〜 0：CODE（パターンコード 0〜255）

プログラム例

```
*******************************************
* BGTEXTCL                            $CC
*******************************************

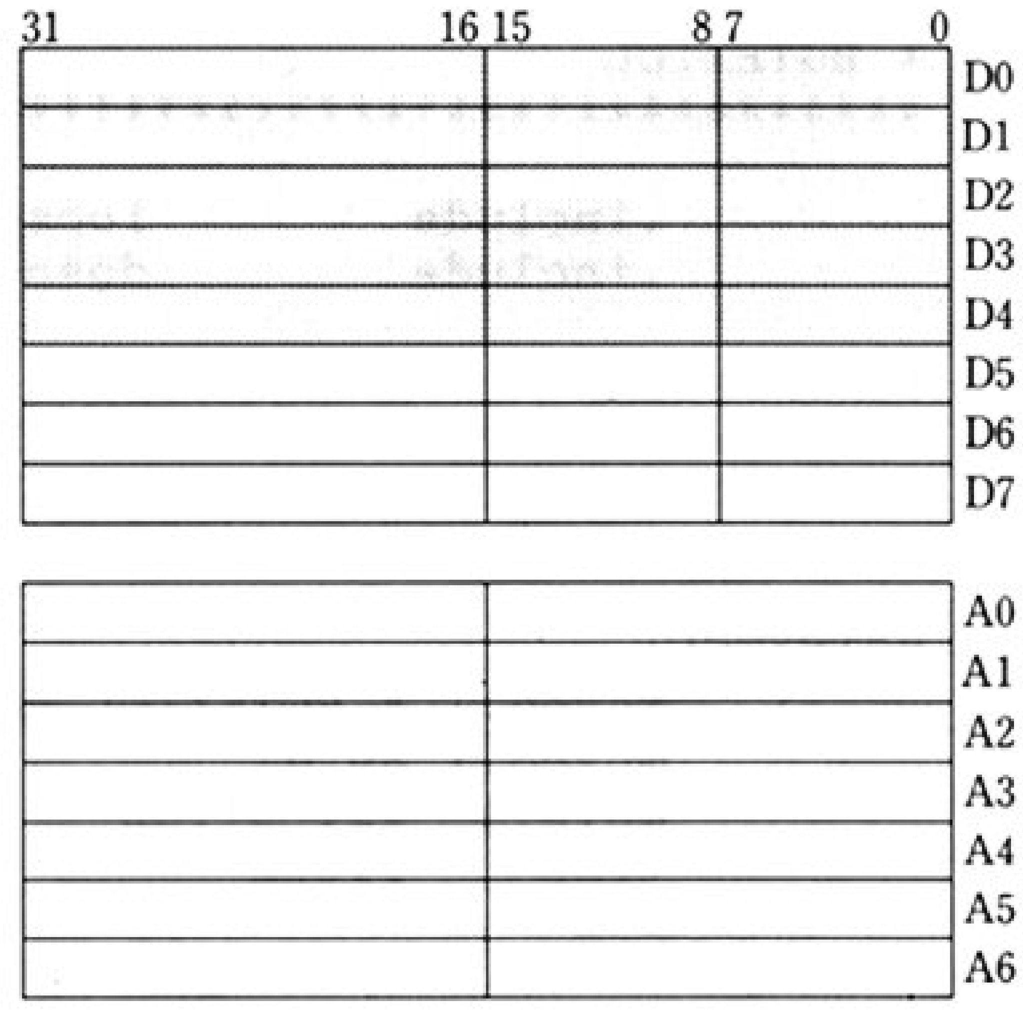
        .include        iocscall.mac
        .include        doscall.mac

iocs    equ     $0f

        .text
        .
        .
BGTEXTCL:
        moveq.l #_BGTEXTCL,d0   * IOCSコール番号をセット
        moveq.l #0,d1           * テキストページ 0
        move.l  #$1_01,d2       * パレットブロック 1、パターンコード $01
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

BGTEXTST

● BGTEXTST　　IOCS コール番号：$CD



機　　能　バックグラウンドテキストの設定

入　　力

D0. L　　$CD　IOCS コール番号
D1. L　　テキストページ
D2. L　　テキスト X 座標
D3. L　　テキスト Y 座標
D4. L　　パターンコード

リターン

D0. L　　終了状態
　D0. L ＝ 0………正常終了
　D0. L ＝ −1……画面モードが不正

解　　説

バックグラウンドのテキストを設定します。
入力の D1. L は 0 ならばテキストページ 0 を、1 ならばテキストページ 1 を指定します。
入力の D2. L、D3. L はそれぞれテキスト X 座標、Y 座標です。
0〜63 の値をとります。
入力の D4. L は、パターンコードです。
パターンコードは、次の 4 つのデータを各ビットに割りあてられたロングワードです。
内容は次の通りです。

ビット 31〜16：常に 0
ビット 15：　V（縦方向）
　0……反転しない
　1……反転する
ビット 14：　H（横方向）
　0……反転しない
　1……反転する
ビット 13〜12：常に 0
ビット 11〜 8：COLOR（パレットブロック 0〜15）
　1 つのパレットに 16 種類の色が指定できます。
　パレットブロックは 16 個あり、そのパレットブロックの番号を指定します。
ビット 7〜 8：CODE（パターンコード 0〜255）

プログラム例

```
*******************************************
* BGTEXTST                             $CD
*******************************************

        .include        iocscall.mac
        .include        doscall.mac

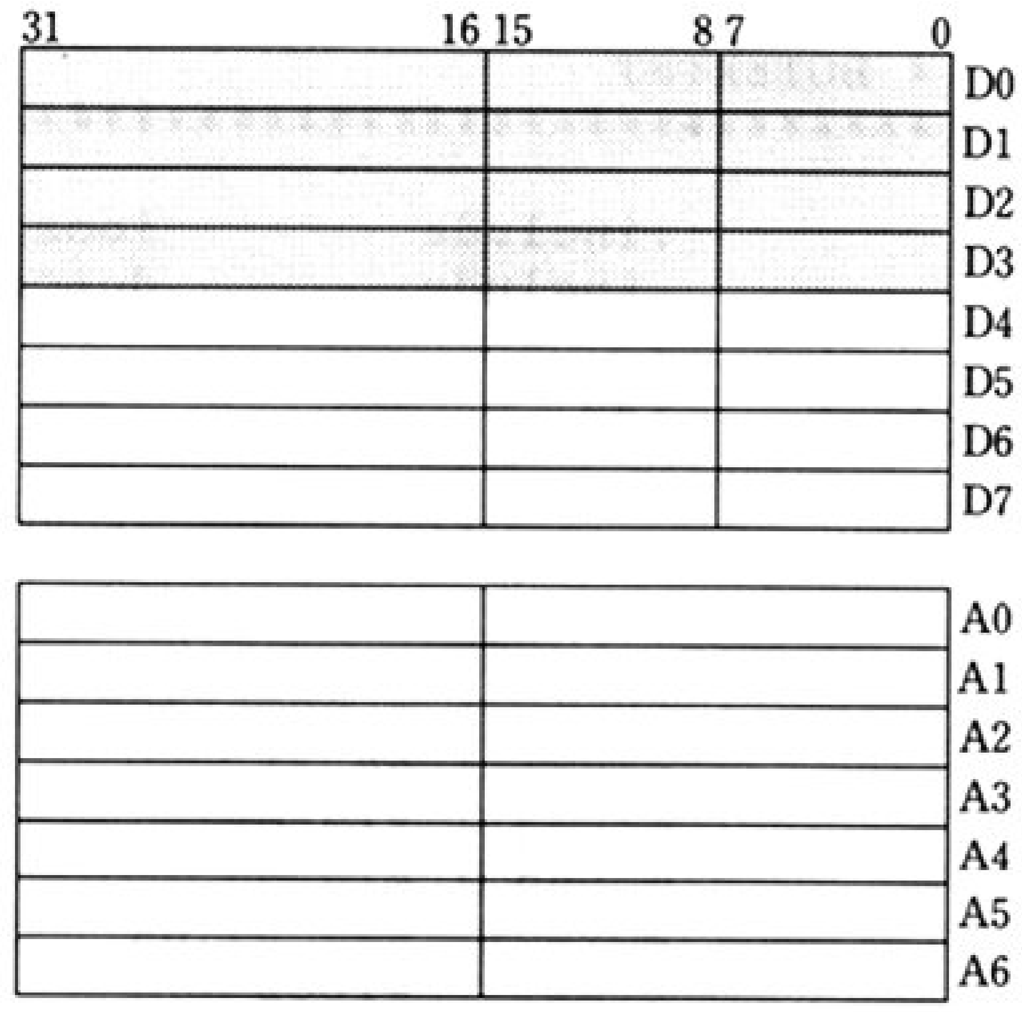
iocs    equ     $0f

        .text
        .
        .
BGTEXTST:
        moveq.l #_BGTEXTST,d0   * IOCSコール番号をセット
        moveq.l #0,d1           * バックグラウンド テキストページ 0
        moveq.l #10,d2          * X 座標 10
        moveq.l #15,d3          * Y 座標 15
        move.l  #$1_01,d4       * パレットブロック 1、パターンコード $01
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

BGTEXTGT

● BGTEXTGT　　IOCSコール番号：$CE

機　能　バックグラウンドテキストの読み出し



入　力

D0. L	$CE	IOCSコール番号
D1. L	テキストページ	
D2. L	テキストX座標	
D3. L	テキストY座標	

リターン　D0. L　　パターンコード

D0. L ＝－1……画面モードが不正

解　説　スプライトのバックグラウンドの、指定した位置のデータを読み込みます。

入力のD1. Lは0ならばテキストページ0を、1ならばテキストページ1を指定します。

入力のD2. L、D3. Lはそれぞれテキスト X座標、Y座標です。

0～63の値をとります。

リターンのD0. Lのパターンコードは、次の4つのデータを各ビットに割りあてられたロングワードです。

内容は次の通りです。

ビット31～16：常に0

ビット15：　V（縦方向）

0……反転していない

1……反転している

ビット14：　H（横方向）

0……反転していない

1……反転している

ビット13～12：常に0

ビット11～ 8：COLOR（パレットブロック0～15）

1つのパレットに16種類の色が指定できます。

パレットブロックは16個あり、そのパレットブロックの番号を指定します。

ビット 7～ 0：CODE（パターンコード0～255）

プログラム例

```
********************************************
* BGTEXTGT                              $CE
********************************************

        .include        iocscall.mac
        .include        doscall.mac

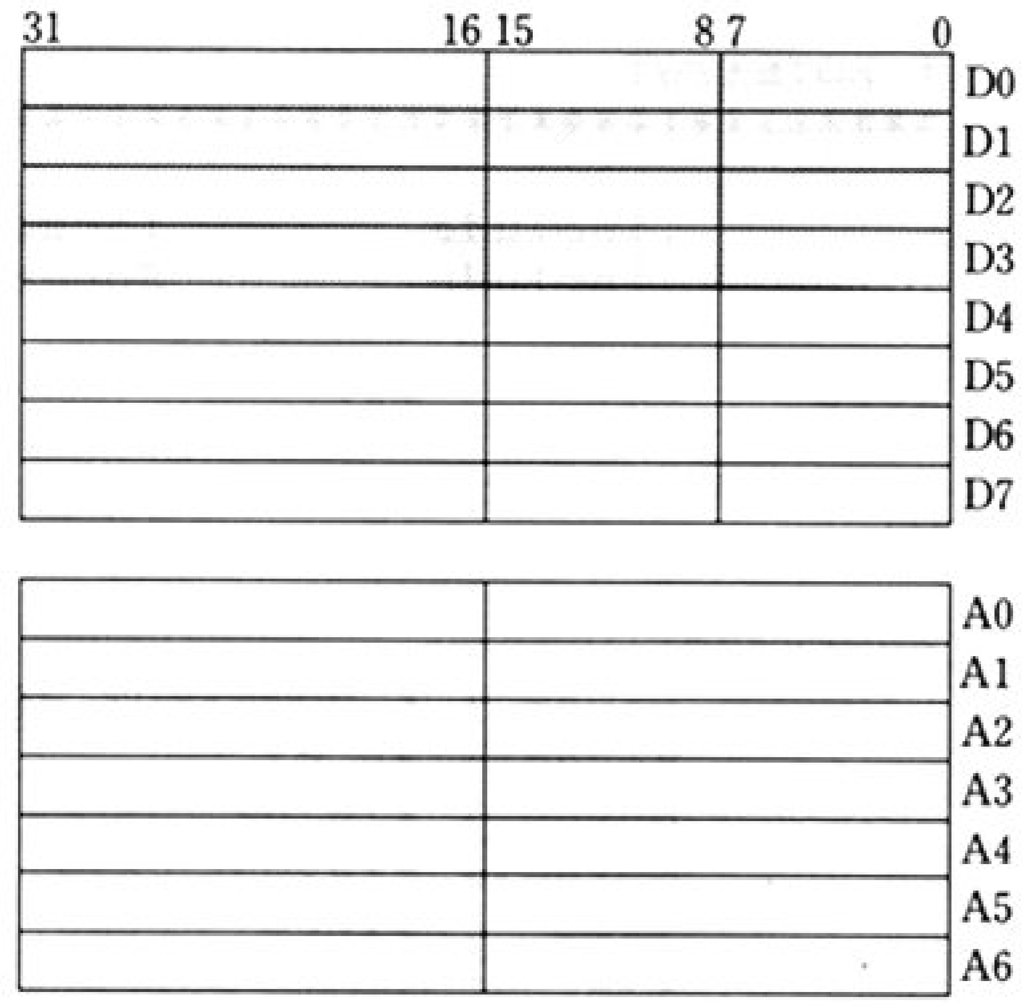
iocs    equ     $0f

        .text
        .
        .
BGTEXTGT:
        moveq.l #_BGTEXTGT,d0   * IOCSコール番号をセット
        moveq.l #0,d1           * バックグラウンド テキストページ 0
        moveq.l #10,d2          * X 座標 10
        moveq.l #15,d3          * Y 座標 15
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

SPALET

● SPALET　　　IOCS コール番号：$CF

機　　能	スプライトパレットの設定／読み出し
入　　力	D0. L　　$CF　　IOCS コール番号 D1. L　　パレットコード、垂直帰線期間の検出の有無 D2. L　　パレットブロック D3. L　　カラーコード
リターン	D0. L　　設定前のカラーコード D0. L ＝－1 ……画面モードが不正 D0. L ＝－2 ……パレットブロック 0 を設定しようとした／読み出そうとした



解　　説

スプライトパレットを、設定したり読み出したりします。
入力の D1. L の内容を次に示します。

ビット 31：垂直帰線期間の検出の有無
　0……検出後設定する
　1……検出せずにすぐ設定する
ビット 3〜0：パレットコード(0〜15)
　1 つのパレットブロックに 16 種類の色が指定でき、その中からどの色についてかを指定します。

入力の D2. L は、パレットブロックです。
パレットブロックは 16 個あり、そのパレットブロックの番号を指定します。
値は 1〜15 で、0 は指定できません。
入力の D3. L は、－1 の場合、読み出しのみで変更されないことを示します。
その他の値の場合はカラーコード(0〜65535)です。
なお、垂直帰線期間を検出せずにパレットを変更すると、画面にちらつきが生じます。

プログラム例

```
********************************************
* SPALET                                 $CF
********************************************

        .include        iocscall.mac
        .include        doscall.mac

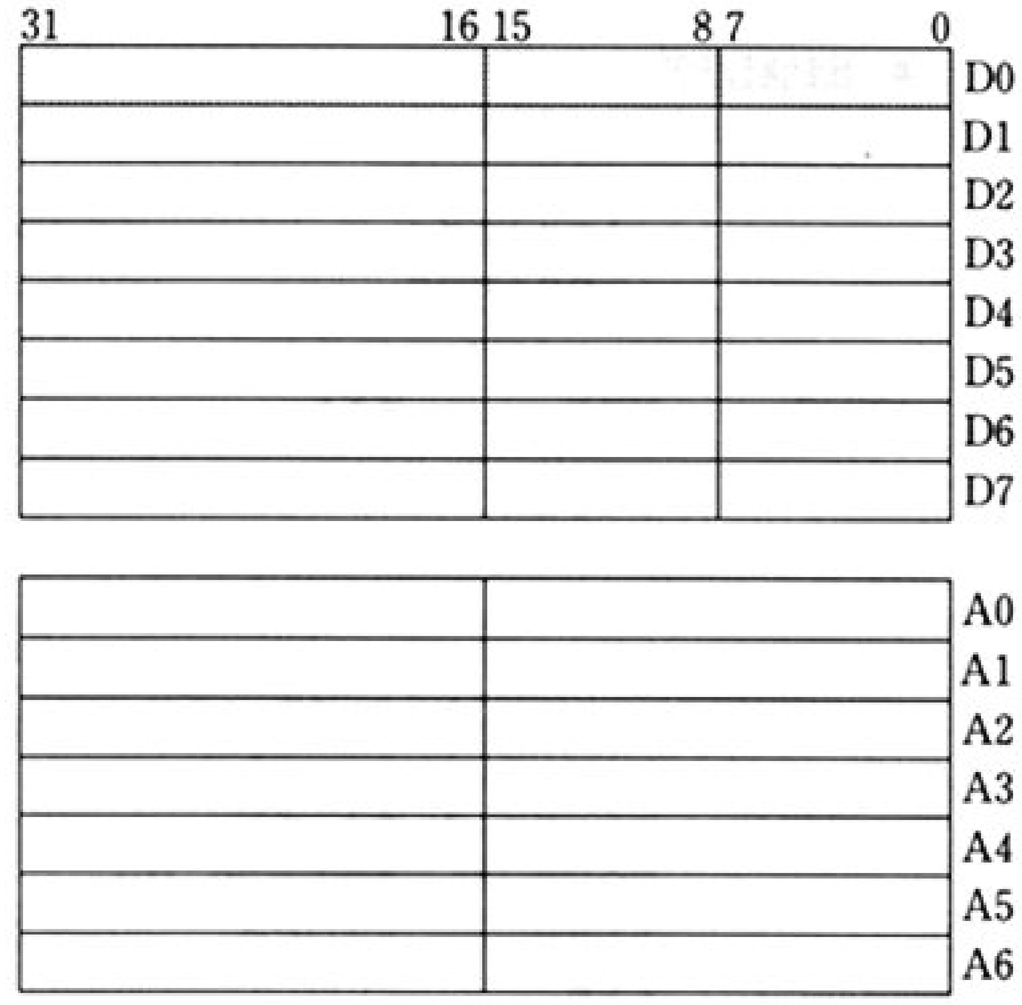
iocs    equ     $0f

        .text
        .
        .
SPALET:
        moveq.l #_SPALET,d0     * IOCSコール番号をセット
        moveq.l #1,d1           * パレットコード 1
        moveq.l #1,d2           * パレットブロック 1
        move.l  #$f83e,d3       * カラーコード
        trap    #iocs           * IOCSコール
        rts
        .
        .
        .end
```

ABORTRST

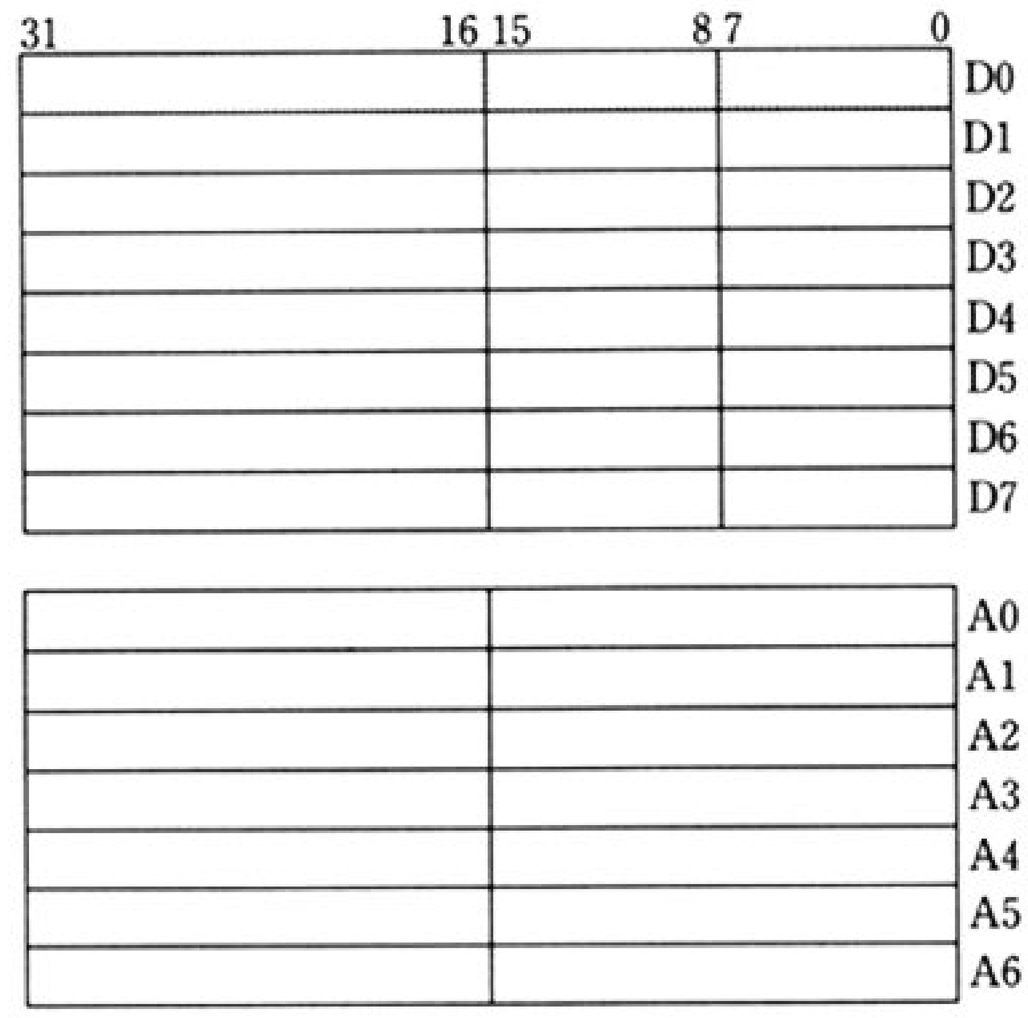
● ABORTRST　　　IOCS コール番号：$FD

機　　能	アボートするための環境の再設定
入　　力	D0. L　　$FD　　IOCS コール番号
リターン	なし（D0. L の値は保証されません）
解　　説	アボートするための環境を再設定します。 OS で使用します。



ABORTJOB

● ABORTJOB　　IOCS コール番号：$FF



機　　能　アボート

入　　力　D0. L　　$FF　　IOCS コール番号

リターン　リターンしない

解　　説　アボートします。
OSで使用します。

第4章

日本語FPのファンクションコール

FPファンクションコールの呼び出し
個々のFPファンクションコール

X68000では、日本語処理（かな漢字変換処理システム）を「日本語FP」と呼び、デバイスドライバ形式で準備しています。
このデバイスドライバの名前は“ASK68K. SYS”といい、CONFIG. SYSファイル内に

DEVICE=¥SYS¥ASK68K. SYS　B：¥X68K_M. DIC　B：X68K_S. DIC ¥ASK¥ENV1. ASK

と記述することによって、起動時にシステム内部に組み込まれ、以降連文節変換によるかな漢字変換処理システムを使用することができるようになります。
日本語FPが組み込まれることによって、日本語FPデバイスドライバ内部のサブルーチンを「日本語FPのファンクションコール」として使用できるようになります。
これらはおもに、

①種々のかな漢字変換モードの管理
②文字種の変換
③変換候補群の管理
④再変換と確定
⑤辞書ファイルの管理

などを行うサブルーチン群です。
日本語入力や、かな漢字変換を利用するようなアプリケーション（ユーザープログラム）は、これらのファンクションコールを用いることによって、かな漢字変換にまつわる管理や作業を統一的に実現することができます。
また、FPコール番号50～56は、日本語FP（ASK68K. SYS）のバージョン2. 00以降のものよりサポートされています。
なお、利用の際は必ずFPコール50番でバージョン番号を確認してください。

4.1 FPファンクションコールの呼び出し

FPファンクションコールは、Human68kのDOSコールを拡張した形で行われます。
すなわち、“ASK68K. SYS”がデバイスドライバとしてシステムに登録されると、DOSコールのknjctrl（$FF22；かな漢字変換用ファンクションコール呼び出し）、keyctrl（$FF24；コンソールから直接入力する）のベクタが書き換えられます。
また、かな漢字変換時の変換用ウィンドウの管理には、同じくDOSコールのhendsp（$FF18；漢字変換行をコントロールする）を使用します。
具体的には次のようにコールします。

●アセンブリ言語の場合

```
move. l    paramn, -(sp)
move. l    paramn-1, -(sp)
                 :
move. l    param2, -(sp)
move. l    param1, -(sp)
dc. w      _KNJCTRL
lea        n *4 (sp)
```

参考：アプリケーションでHuman68kの漢字変換ウィンドウhendsp（$FF18）を使いたくないときの処理は次の2通りの方法が考えられます。

①hendsp（$FF18）をアプリケーションでエミュレートする
②knjctrl（$FF22）でFPのかな漢字変換カーネルを呼び出す

4.2 個々のFPファンクションコール

それでは、個々のFPファンクションコールについて解説します。
解説の方法は第2章と同じです。

FPコール番号：1

機　　能　かな漢字変換モードに入ります／出ます

引　　数　MODE　かな漢字変換モードの設定
= 0 ……かな漢字変換なし（通常の入力モード）
= 1 ……一括変換、辞書の先読みなし
= 2 ……一括変換、辞書の先読みあり
= 3 ……逐次変換

コ ー ル

```
MOVE. L    MODE, －(SP)
MOVE. L    #1, －(SP)
DC. W      _KNJCTRL
ADDQ. L    #8, SP
```

リターン　0またはエラーコード
= 0……正常終了
= 1……ドライブの準備ができていません
= 2……モードの誤り

解　　説　ユーザーがキーボードから[CTRL]+[XF1]キーを押したのと同じように、かな漢字変換モードに入ったり出たりします。
MODE＝1, 2, 3のときには、画面の最下行にかな漢字変換用のウィンドウが表示されます。
以降のキーボードの入力は、かな漢字変換ウィンドウで処理されます。
MODE＝0のときには、かな漢字変換ウィンドウがクローズされます。

プログラム例

```
move. l    #2, －(sp)
move. l    #1, －(sp)
dc. w      _KNJCTRL      *一括変換、辞書先読みモードに入る
addq. l    #8, SP        *以降はかな漢字変換をしながらキーボード入力する
           ⋮
clr. l     －(sp)
move. l    #1, －(sp)
dc. w      _KNJCTRL      *通常入力モードに戻る
addq. l    #8, sp
```

FPコール番号：2

機　能　かな漢字モードの通知

引　数　なし

コール

```
MOVE.L      #2,-(SP)
DC.W        _KNJCTRL
ADDQ.L      #4,SP
```

リターン　かな漢字モード（モードの値はFPコール番号1のMODEと同じ）

解　説　日本語FPの現在のかな漢字変換モードを通知します。

プログラム例

```
move.l      #2,-(sp)
dc.w        _KNJCTRL
addq.l      #4,sp           * D0レジスタに現在のモードが返る
```

FPコール番号：3

機　　能　フラッシュモードの設定

引　　数　MODE　　フラッシュモード
　　　　　　　　＝0　……フラッシュモードの解除
　　　　　　　　＝1　……フラッシュモードの設定

コ ー ル

```
MOVE. L    MODE, −(SP)
MOVE. L    #3, −(SP)
DC. W      _KNJCTRL
ADDQ. L    #8, SP
```

リ タ ー ン　0またはエラーコード
　＝0……正常終了
　＝2……モード指定の誤り

解　　説　フラッシュモード（日本語FPのキーボードバッファを空にするかどうかのモード）を設定します。

プログラム例

```
move. l    #1, −(sp)
move. l    #3, −(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          ＊フラッシュつきキー入力時にFPのバッファをクリアする
           ⋮
clr. l     −(sp)
move. l    #3, −(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          ＊フラッシュつきキー入力時にFPのバッファをクリアしない
```

FPコール番号：4

機　能	フラッシュモードの通知
引　数	なし
コール	MOVE. L　#4, −(SP) DC. W　_KNJCTRL ADDQ. L　#4, SP
リターン	現在のフラッシュモード（モードの値はFPコール番号3のMODEと同じ）
解　説	現在のフラッシュモードの状態を通知します。

プログラム例

```
move. l    #4, −(sp)
dc. w      _KNJCTRL
addq. l    #4, sp          * D0レジスタに現在のフラッシュモードが返る
```

FPコール番号：5

機　能　入力モードの設定

引　数　MODE　入力モード

	D31	～	D03	D02	D01	D00
MODE	0	0	0			

- D00
  - 0：通常モード
  - 1：ローマ字変換
- D01
  - 0：カタカナ
  - 1：ひらがな
- D02
  - 0：半角
  - 1：全角

コール

```
MOVE. L    MODE, －(SP)
MOVE. L    #5, －(SP)
DC. W      _KNJCTRL
ADDQ. L    #8, SP
```

リターン　0またはエラーコード

＝ 0……正常終了

＝ 2……モード指定の誤り

解　説　MODEのビットパターンによって、FPの入力モードを設定します。

FPコール番号：5

プログラム例

```
move. l     #7, -(sp)
move. l     #5, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *ローマ字変換、全角、ひらがなモードに設定
move. l     #2, -(sp)
move. l     #1, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *一括変換、辞書先読みモードでかな漢字変換モードに入る
            ⋮
            ⋮             *キーボード入力、かな漢字変換
            ⋮
clr. l      -(sp)
move. l     #1, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *通常入力モードに戻る
```

FPコール番号：6

機　能　入力モードの通知

引　数　なし

コール

```
MOVE. L     #6, －(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　入力モード（値はFPコール番号5のMODEと同じ）

解　説　日本語FPの現在の入力モードを通知します。

プログラム例

```
move. l     #6, －(sp)
dc. w       _KNJCTRL
addq. l     #4, sp          ＊D0 レジスタに現在の入力モードが返る
```

FPコール番号：7

機　　能　かな漢字変換モードのロック

引　　数　MODE　　かな漢字変換モードをロックするか否か
　　　　　　　　　　＝ 0　……かな漢字変換モードをロック
　　　　　　　　　　＝ 1　……かな漢字変換モードをアンロック

コ ー ル

```
MOVE. L     MODE, －(SP)
MOVE. L     #7, －(SP)
DC. W       _KNJCTRL
ADDQ. L     #8, SP
```

リターン　0またはエラーコード
　＝ 0……正常終了
　＝ 2……モード指定の誤り

解　　説　かな漢字変換モードや入力モードをキーボードから変更することを可能にしたり不可能にしたりします。

FPコール番号：７

プログラム例

```
move. l     #0, -(sp)
move. l     #7, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *かな漢字変換モードをロック状態にする
move. l     #1, -(sp)
move. l     #1, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *一括変換、辞書先読みなしモードでかな漢字変換モードに
                           入る
            :             *キーボード入力、かな漢字変換
            :
move. l     #0, -(sp)
move. l     #1, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *通常入力モードに戻す
move. l     #1, -(sp)
move. l     #7, -(sp)
dc. w       _KNJCTRL
addq. l     #8, sp        *モードロック状態を解除する
```

FPコール番号：8

機　　能　かな漢字変換モードロック状態の通知

引　　数　なし

コ ー ル

```
MOVE. L     #8, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　現在のロック状態（値はFPコール番号7のMODEと同じ）

解　　説　日本語FPの現在のかな漢字変換モードのロック状態を通知します。

プログラム例

```
move. l     #8, -(sp)
dc. w       _KNJCTRL
addq. l     #4, sp          * D0レジスタに現在のFPのかな漢字変換モードのロック
                              状態が返る
```

FPコール番号：9

機　能　JIS／句点モードの設定

引　数

MODE	JIS／句点モードの指定
	＝ 0……JIS／シフト JIS モード
	＝ 1……句点モード

コール

```
MOVE. L     MODE, －(SP)
MOVE. L     #9, －(SP)
DC. W       _KNJCTRL
ADDQ. L     #8, SP
```

リターン　0またはエラーコード
＝ 0……正常終了
＝ 2……モード指定の誤り

解　説　かな漢字変換モードでのコード入力のコード体系を指定します。

プログラム例

```
move. l     #1, －(sp)
move. l     #9, －(sp)
dc. w       _KNJCTRL
addq. l     #8, sp          ＊コード入力時のモードに句点モードを選択
```

FPコール番号：10

機　能　JIS／句点モードの通知

引　数　なし

コール

```
MOVE. L     #10, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　コード入力時のモード

　= 0……JIS／シフト JIS モード

　= 1……句点モード

解　説　日本語 FP の現在のコード入力のコード体系を通知します。

プログラム例

```
move. l     #10, -(sp)
dc. w       _KNJCTRL
addq. l     #4, sp          *D0レジスタに現在のFPのコード入力のコード体系が返る
```

FPコール番号：11

機　　能　学習モードの設定

引　　数　MODE　学習モード

= 0……一時学習モード（メモリに学習）

= 1……辞書更新学習モード（ディスクに学習）

コ ー ル

```
MOVE. L    MODE, −(SP)
MOVE. L    #11, −(SP)
DC. W      _KNJCTRL
ADDQ. L    #8, SP
```

リターン　0またはエラーコード

= 0……正常終了

= 2……モード指定の誤り

解　　説　日本語FPの学習モードを設定します。

プログラム例

```
move. l    #1, -(sp)
move. l    #11, -(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          *辞書更新学習モードにする
move. l    #1, -(sp)
move. l    #1, -(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          *一括変換、辞書先読みなしモードでかな漢字変換モードに
                            入る
           ⋮               *キーボード入力、かな漢字変換
           ⋮
move. l    #0, -(sp)
move. l    #1, -(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          *通常入力モードに戻る
move. l    #0, -(sp)
move. l    #11, -(sp)
dc. w      _KNJCTRL
addq. l    #8, sp          *一時学習モードにする
```

FPコール番号：12

機　　能　学習モードの通知

引　　数　なし

コ ー ル

```
MOVE. L     #12, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　学習モード（値は FP コール番号 11 の MODE と同じ）

解　　説　日本語 FP の現在の学習モードを通知します。

プログラム例

```
move. l     #12, -(sp)
dc. w       _KNJCTRL
addq. l     #4, sp          * D0 レジスタに現在の FP の学習モードが返る
```

FPコール番号：13〜18

機能　文字列の変換

引数　FNO　変換方法（コール番号）

= 13 ……半角文字列 → 全角文字列
= 14 ……全角文字列 → 半角文字列
= 15 ……半角文字列 → 全角カタカナ文字列（ローマ字変換）
= 16 ……半角文字列 → 全角ひらがな文字列（ローマ字変換）
= 17 ……全角カタカナ文字列 → 全角ひらがな文字列
= 18 ……全角ひらがな文字列 → 全角カタカナ文字列

SOURCE　変換前の文字列の格納アドレス
DEST　変換後の文字列を格納するアドレス

コール

```
PEA       DEST
PEA       SOURCE
MOVE. L   FNO, -(SP)
DC. W     _KNJCTRL
LEA       12(SP), SP
```

リターン　0またはエラーの原因になった文字の位置

= 0 ……正常終了
≠ 0 ……変換エラーが起きた文字の、バッファの先頭からの文字数（半角文字計算）

解説　日本語FPの現在のモードにかかわらず、かな漢字変換用サブルーチンを呼び出して文字列のさまざまな変換を行います。
変換前後の文字列は79バイト以下とし、各文字列はヌル文字（0）で終了しているものとします。

FPコール番号：13〜18

プログラム例

```
pea         dest
pea         source
move. l     #16, -(sp)
dc. w       _KNJCTRL
lea         12 (sp), sp      *半角文字列をローマ字と見なし、ローマ字かな変換をして
                              全角ひらがな文字列として返る
 :                           * dest には 'かんじ', 0 が入り、D0 レジスタには 0 が返る
source :
    dc. b   'kanji', 0
dest :
    ds. b    80
```

FPコール番号：19

機　能　一括変換、および逐次先読みの、最初の変換候補群と後続文節の作成

引　数

BUF　変換する全角ひらがな文字列のアドレス
KOUHO　最初の候補の格納バッファ
KOUUOKU 後続文節の格納バッファ

コール

```
PEA       KOUZOKU
PEA       KOUHO
PEA       BUF
MOVE. L   #19, -(SP)
DC. W     _KNJCTRL
LEA       16 (SP), SP
```

リターン　候補数

解　説

BUFの値が0のとき、逐次先読みと解釈され、すでに送られている文字列を変換します。

FPコール番号19〜27、33〜40までは、かな漢字変換モジュールのサブルーチンを呼び出し、かな漢字変換を行います。

これらのコールを使用するアプリケーションは、FPを通常入力モードに設定し、モードロック状態にして、自らキーボード入力を受け付け、入力文字列をFPコール番号13〜18で全角ひらがな文字列に変換した後、これらのサブルーチンを呼び出さなければなりません。

これらのサブルーチンの入力文字列は、全角ひらがな文字列だけです（全角ひらがな文字以外は漢字変換の対象になりません）。

FPコール番号：19

プログラム例

```
clr.l       -(sp)
move.l      #7, -(sp)
dc.w        _KNJCTRL
addq.l      #8, sp          *モードロック
clr.l       -(sp)
move.l      #1, -(sp)
dc.w        _KNJCTRL
addq.l      #8, sp          *通常入力状態
            :
            :               *キーボード入力、FPコール番号13～18による全角ひら
                             がな文字列への変換
pea         kouzoku         *後続文字列の格納バッファ
pea         kouho           *候補の格納バッファ
pea         buf             *全角ひらがな文字列
move.l      #19, -(sp)
dc.w        _KNJCTRL
lea         16(sp), sp      *一括変換
            :               *FPコール番号20、21により、次および前の候補と後続
                             文節を得る
            :               *現候補ブロックに適当な候補がないときは、FPコール番
                             号22、23で他のブロックを得る
            :               *必要に応じて、FPコール番号33、34で先頭文節の長さ
                             を変える
            :
pea         kouho
move.l      #3, -(sp)
move.l      #25, -(sp)
dc.w        _KNJCTRL
lea         12(sp)          *候補番号の3で先頭文節を確定
            :               *D0レジスタに、確定した文節の読みの長さが返る
            :
```

⋮	＊後続文節をさらに変換する場合は、D0 レジスタの値をもとに後続文節の読みの先頭の位置を求め、FP コール番号 19 からやり直す
⋮	＊ FP コール番号 24 で全体を確定する

FPコール番号：20

機　　能　候補群ブロック内の前候補を得ます

引　　数

KOUHO	候補の格納バッファ
KOUZOKU	後続文節の格納バッファ

コ ー ル

```
PEA      KOUZOKU
PEA      KOUHO
MOVE. L  #20, -(SP)
DC. W    _KNJCTRL
LEA      12(SP), SP
```

リターン　0または候補番号

　＝0　……候補なし
　≠0　……候補番号

解　　説　現在の候補群のブロックの中で、前候補となる語の候補番号を得ます。
FP コール番号 19 の解説をご覧ください。

FPコール番号：21

機　　能	候補群ブロック内の次候補を得ます
引　　数	KOUHO　　候補の格納バッファ KOUZOKU　後続文節の格納バッファ

コール

```
PEA       KOUZOKU
PEA       KOUHO
MOVE.L    #21,-(SP)
DC.W      _KNJCTRL
LEA       12(SP),SP
```

リターン　0または候補番号

　＝ 0　……候補なし

　≠ 0　……候補番号

解　　説　現在の候補群のブロックの中で、次候補となる語の候補番号を得ます。

FP コール番号 19 の解説をご覧ください。

FPコール番号：22

機　能　前候補ブロックの作成

引　数

KOUHO	最初の候補の格納バッファ
KOUZOKU	後続文節の格納バッファ

コール

```
PEA       KOUZOKU
PEA       KOUHO
MOVE.L    #22, -(SP)
DC.W      _KNJCTRL
LEA       12(SP), SP
```

リターン　0または候補数

＝0　……候補ブロックなし

≠0　……候補番号

解　説　前候補ブロックを作成します。

FPコール番号19の解説をご覧ください。

FPコール番号：23

機　　能	次候補ブロックの作成	
引　　数	KOUHO	最初の候補の格納バッファ
	KOUZOKU	後続文節の格納バッファ
コ ー ル	PEA	KOUZOKU
	PEA	KOUHO
	MOVE. L	#23, −(SP)
	DC. W	_KNJCTRL
	LEA	12 (SP), SP
リターン	0または候補数	
	= 0 ……候補ブロックなし	
	≠ 0 ……候補数	
解　　説	次候補ブロックを作成します。 FP コール番号 19 の解説をご覧ください。	

FPコール番号：24

機　　能　全体確定

引　　数　BUF　変換後の文字列を格納するバッファ

コ ー ル

```
PEA       BUF
MOVE. L   #24, -(SP)
DC. W     _KNJCTRL
ADDQ. L   #8, SP
```

リターン　0 またはエラーコード

＝ 0 ……正常終了

≠ 0 ……エラー

解　　説

全体を確定します。

変換操作の最後に、このコールを行います。

BUF には最初から最後までのすべての文節が格納されます。

また、確定した文節の学習はこの時点で記憶されます。

FP コール番号 19 の解説をご覧ください。

FPコール番号：25

機　　能	先頭文節の確定

引　　数

NO	候補ブロック内での番号（1～?）
BUF	確定する先頭文節の格納バッファ
KOUHO	次文節の最初の候補の格納バッファ
KOUZOKU	次文節に続く後続文節の格納バッファ
NKOUHO	次文節の候補数の格納バッファ（結果は short）

コ　ー　ル

```
PEA         NKOUHO
PEA         KOUZOKU
PEA         KOUHO
PEA         BUF
MOVE. L     NO, -(SP)
MOVE. L     #25, -(SP)
DC. W       _KNJCTRL
LEA         24 (SP), SP
```

リターン　確定した文節の読みの長さ（半角換算値）

＝0 ……候補番号が不正

≠0 ……確定文節の読みの長さ（半角換算値）

解　　説　先頭の文節を確定します。

FP コール番号 19 の解説をご覧ください。

FPコール番号：26

機　　能　逐次変換および先読み変換のための文字列の追加

引　　数

BUF	追加する文字列のアドレス
RETBUF	途中の変換文字列を格納するバッファ（ただし先読みモードではBUFの内容がそのままコピーされます）
MODE	変換タイプのモード ＝ 0……逐次変換 ＝ 1……先読み変換 逐次変換と先読み変換の違いは、漢字に変換した中間結果を返すかどうかだけです。

コ ー ル

```
MOVE. L    MODE, －(SP)
PEA        RETBUF
PEA        BUF
MOVE. L    #26, －(SP)
DC. W      _KNJCTRL
LEA        16 (SP), SP
```

リターン　0またはエラーコード

＝ 0　……正常終了

≠ 0　……エラー（文字列追加のオーバーフローなど）

解　　説　詳細はFPコール番号27の解説をご覧ください。

BUFの値が0のとき、逐次先読みが初期化されます。

このとき、過去に追加された文字列はキャンセルされます。

FPコール番号：27

機能　逐次変換および先読み変換のための文字の削除

引数

N　最後から削除する文字列のバイト数

RETBUF　途中の変換文字列を格納するバッファ
ただし先読みモードでは、現在の読みの内容がそのままコピーされます。

MODE　変換タイプのモード
= 0……逐次変換
= 1……先読み変換

コール

```
MOVE. L     MODE, -(SP)
PEA         RETBUF
MOVE. L     N, -(SP)
MOVE. L     #27, -(SP)
DC. W       _KNJCTRL
LEA         16 (SP), SP
```

リターン　0またはエラーコード
= 0 ……正常終了
≠ 0 ……エラー

解説

FPコール番号26, 27は、逐次変換および辞書先読み一括変換モードのためのコールで、FPコール番号19に対応しています。
FPコール番号19では、わたされた全角ひらがな文字列そのものを変換対象としますが、FPコール番号26、27ではわたされた文字列をバッファに追加（または削除）し、辞書を読んで変換しなおします。
変換された候補と後続文節のとり出しには、FPコール番号20〜23、33、34を用い、確定にはFPコール番号24、25を使います。
確定された読みは、変換用バッファから除かれます。
逐次変換と辞書先読み一括変換モードの区別は、アプリケーションの表示タイミングによります。
FPコール番号19をコールすると、一括、先読み、逐次のいずれの場合も候補選択の状態になります。
各文節の候補を選択し、必要ならFPコール番号35、36で文節を前後に動かし、最後にFPコール番号24で全体を確定してください。
このとき、全文節が吐き出されます。
文節の候補を最初から順に決定していくときでも、最後にFPコール番号24をコールしてください。

FPコール番号：28

機　　能　辞書のオープン

引　　数　なし

コ ー ル

```
MOVE. L      # 28, －(SP)
DC. W        _KNJCTRL
ADDQ. L      # 4, SP
```

リターン　0 またはエラーコード

＝ 0　……正常終了

≠ 0　……エラー（＝ 1 は「ドライブの準備ができていない」）

解　　説　辞書ファイルをオープンします。

これにより、FP コール番号 19～27、30、31、33～40 が動作するようになります。

プログラム例

```
move. l      # 28, －(sp)
dc. w        _KNJCTRL
addq. l      # 4, sp         ＊辞書のオープン
```

FPコール番号：29

機　能	辞書のクローズ
引　数	なし
コール	MOVE. L　#29, −(SP) DC. W　_KNJCTRL ADDQ. L　#4, SP
リターン	0またはエラーコード ＝0　……正常終了 ≠0　……エラー
解　説	辞書ファイルをクローズします。 この後、FPコール番号19〜27、30、31、33〜40は動作しません。

プログラム例

```
move. l     #29, −(sp)
dc. w       _KNJCTRL
addq. l     #4, sp          *辞書のクローズ
```

FPコール番号：30

機　　能　辞書への単語登録

引　　数

FD　　辞書の種類
　　＝ 0……メイン辞書
　　＝ 1……サブ辞書
YOMI　　登録する単語の読み（全角ひらがな、英数字）
TANGO　登録する単語の文字列
SYN　　文法コード（1～30）（次頁表を参照）

コ ー ル

```
MOVE. L    SYN, -(SP)
PEA        TANGO
PEA        YOMI
MOVE. L    FD, -(SP)
MOVE. L    #30, -(SP)
DC. W      _KNJCTRL
LEA        20 (SP), SP
```

リターン　0またはエラーコード

＝ 0　……正常終了
≠ 0　……エラー（登録できなかった）
　＝ 1……辞書の種類が不適当
　＝ 2……文法コードが不適当
　＝ 3……読みが不適当
　＝ 4……登録するページがない

解　　説　指定した辞書に、指定した単語を文法コードつきで登録します。
この前に、FPコール番号７でかな漢字変換モードのロックをしておいてください。

プログラム例

```
    move.l  #20, -(sp)      *人名（名）
    pea     tango
    pea     yomi
    move.l  #1, -(sp)
    move.l  #30, -(sp)
    dc.w    _KNJCTRL
    lea     20(sp), sp      *サブ辞書に単語登録
       ⋮
yomi:
    dc.b    'しんぺい', 0
tango:
    dc.b    '真平', 0
```

FPコール番号：30

コード	品詞の種類	品　詞　の　例
0	―――	
1	動詞カ行五段活用	聞－く
2	動詞ガ行五段活用	泳－ぐ
3	動詞サ行五段活用	話－す
4	動詞タ行五段活用	勝－つ
5	動詞ナ行五段活用	死－ぬ
6	動詞バ行五段活用	飛－ぶ
7	動詞マ行五段活用	読－む
8	動詞ラ行五段活用	切－る
9	動詞ワ行五段活用	使－う
10	動詞サ行変格活用	する
11	動詞カ行変格活用	来る
12	動詞一段活用	見－る　教え－る（上一段活用、下一段活用）
13	形容詞	新し－い　良－い（ク活用とシク活用は区別しない）
14	形容動詞	静か－だ
15	形容動詞複合名詞	純粋（な人）のように形容動詞としても使う名詞
16	サ変複合名詞	愛（する）、惨敗（する）ようにサ変複合動詞になる名詞
17	名詞	形容動詞の語幹やサ変複合動詞にならない名詞
18	単漢字	漢字の読みの登録
19	人名（姓）	人名の姓
20	人名（名）	人名の名
21	地名	地名
22	団体名	会社名、法人名、政党名など
23	物の名称	商品名など
24	数詞	個、匹など
25	数字	数字
26	接尾語	語の下につける語（さ、さん、めく、ぽい、がるなど）
27	感動詞	感動や呼びかけ、話の受け答え（おや、まあ、はいなど）
28	接続詞	前の言葉と後の言葉をつなぐ語（または、そこでなど）
29	副詞	活用語を修飾する語（はっきり、たいへん、実になど）
30	連体詞	非活用語を修飾する語（たった、あらゆる、あのなど）

FPコール番号：31

機　能　辞書からの単語削除

引　数

FD　辞書の種類
　= 0……メイン辞書
　= 1……サブ辞書
YOMI　削除する単語の読み（全角ひらがな、英数字）
TANGO　削除する単語の文字列
SYN　文法コード（1〜30）（前頁表を参照）

コール

```
MOVE.L   SYN,-(SP)
PEA      TANGO
PEA      YOMI
MOVE.L   FD,-(SP)
MOVE.L   #31,-(SP)
DC.W     _KNJCTRL
LEA      20(SP),SP
```

リターン　0またはエラーコード

= 0　……正常終了
≠ 0　……エラー（削除できなかった）
　= −1……削除すべき単語が辞書にない
　= 1………辞書の種類が不適当
　= 2………文法コードが不適当
　= 3………読みが不適当
　= 4………辞書がなかった

解　説　指定した辞書から、指定した単語を文法コード参照で削除します。
この前に、FPコール番号7でかな漢字変換モードのロックをしておいてください。

FPコール番号：31

プログラム例

```
        move.l   #19, -(sp)       *人名（姓）
        pea      tango
        pea      yomi
        move.l   #0, -(sp)
        move.l   #31, -(sp)
        dc.w     _KNJCTRL
        lea      20(sp), sp       *メイン辞書から単語を削除
          :
yomi:
        dc.b     'やまぐち', 0
tango:
        dc.b     '山口', 0
```

FPコール番号：32

機　　能　辞書ファイル名の登録

引　　数　MAINDIC　メイン辞書名
SUBDIC　サブ辞書名

コ ー ル

```
PEA         SUBDIC
PEA         MAINDIC
MOVE. L     #32, -(SP)
DC. W       _KNJCTRL
LEA         12 (SP), SP
```

リターン　0またはエラーコード
＝0　……正常終了
＝1　……ドライブの準備ができていません

解　　説　メイン辞書、サブ辞書のファイル名を登録します。

プログラム例

```
        pea       subdic
        pea       maindic
        move. l   #32, -(sp)
        dc. w     _KNJCTRL
        lea       12 (sp), sp              *辞書ファイル名の登録
         :
subdic :
        dc. b     '¥X68K_S. DIC', 0
maindic :
        dc. b     '¥X68K_M. DIC', 0
```

FPコール番号：33

機　能　先頭文節を 1 文字長くして再変換

引　数　KOUHO　最初の候補の格納バッファ
KOUZOKU　後続文節の格納バッファ

コール

```
PEA       KOUZOKU
PEA       KOUHO
MOVE. L   #33, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　候補数

解　説　先頭の文節を読み 1 文字分だけ長くして、再変換します。
FP コール番号 19 の解説をご覧ください。

FPコール番号：34

機　能	先頭文節を1文字短くして再変換
引　数	KOUHO　　最初の候補の格納バッファ KOUZOKU　後続文節の格納バッファ
コール	<pre>PEA KOUZOKU PEA KOUHO MOVE. L #34, −(SP) DC. W _KNJCTRL LEA 12 (SP), SP</pre>
リターン	候補数
解　説	先頭の文節を読み1文字分だけ短くして、再変換します。 FP コール番号 19 の解説をご覧ください。

FPコール番号：35

機　能　前文節への移動

引　数

KOUHO　　前文節の最初の候補の格納バッファ
KOUZOKU　前文節に続く後続文節の格納バッファ

コール

```
PEA       KOUZOKU
PEA       KOUHO
MOVE. L  #35, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　0 または前文節の候補数

　＝ 0　……前文節がありません
　≠ 0　……前文節の候補数

解　説　前の文節に移動します。

FP コール番号 19 の解説をご覧ください。

FPコール番号：36

機　　能	次文節への移動
引　　数	KOUHO　次文節の最初の候補の格納バッファ KOUZOKU　次文節に続く後続文節の格納バッファ

コ ー ル

```
PEA       KOUZOKU
PEA       KOUHO
MOVE. L   #36, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　0または次文節の候補数

　＝ 0　……次文節がありません

　≠ 0　……次文節の候補数

解　　説　次の文節に移動します。

FPコール番号19の解説をご覧ください。

FPコール番号：37

機　　能　先頭文節をカタカナにします

引　　数　KOUHO　　候補の格納バッファ
KOUZOKU　後続文節の格納バッファ

コ ー ル

```
PEA       KOUZOKU
PEA       KOUHO
MOVE. L   #37, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　0またはエラーコード
　＝0　……正常終了
　≠0　……エラー

解　　説　先頭の文節をカタカナに変換します。
FPコール番号37～40までは、候補を選択するのと同じタイミングで使用します。
これらの機能が呼ばれると、最初の候補群ブロックが作成され、カタカナ（あるいは、ひらがな、半角、全角）候補が最初の候補になります。

FPコール番号：38

機　能	先頭文節をひらがなにします
引　数	KOUHO　候補の格納バッファ KOUZOKU 後続文節の格納バッファ

コール

```
PEA      KOUZOKU
PEA      KOUHO
MOVE.L   #38,-(SP)
DC.W     _KNJCTRL
LEA      12(SP),SP
```

リターン　0またはエラーコード

　＝0　……正常終了
　≠0　……エラー

解　説　先頭の文節をひらがなに変換します。
詳細はFPコール番号37の解説をご覧ください。

FPコール番号：39

機　　能　先頭文節を半角文字にします

引　　数　KOUHO　　候補の格納バッファ
KOUZOKU　後続文節の格納バッファ

コ ー ル

```
PEA       KOUZOKU
PEA       KOUHO
MOVE. L   #39, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　0またはエラーコード
　＝0　……正常終了
　≠0　……エラー

解　　説　先頭の文節を半角文字に変換します。
詳細はFPコール番号37の解説をご覧ください。

FPコール番号：40

機　　能　先頭文節を全角文字にします

引　　数　KOUHO　　候補の格納バッファ
KOUZOKU　後続文節の格納バッファ

コ ー ル

```
PEA        KOUZOKU
PEA        KOUHO
MOVE. L    #40, -(SP)
DC. W      _KNJCTRL
LEA        12 (SP), SP
```

リターン　0またはエラーコード
＝ 0 ……正常終了
≠ 0 ……エラー

解　　説　先頭の文節を全角文字に変換します。
詳細はFPコール番号37の解説をご覧ください。

FPコール番号：41

機　　能　使用中の辞書ファイル名の参照

引　　数
MAINDIC　メイン辞書名の文字列格納バッファ
SUBDIC　サブ辞書名の文字列格納バッファ

コール

```
PEA       SUBDIC
PEA       MAINDIC
MOVE. L   #41, -(SP)
DC. W     _KNJCTRL
LEA       12 (SP), SP
```

リターン　0またはエラーコード
　= 0　……正常終了
　≠ 0　……エラー

解　　説　現在使用中のメイン辞書ファイル、サブ辞書ファイルのファイル名を得ます。

FPコール番号：50

機　　能　日本語FP（ASK68K. SYS）のバージョン番号を通知します。

引　　数　なし

コ ー ル

```
MOVE. L     #50, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　バージョン番号またはエラーコード

= 0……エラー

> 0……バージョン番号

= −1……バージョン2. 00以前

解　　説　日本語FP（ASK68K. SYS）のバージョン番号を通知します。

例えばバージョンが2.01の場合は、リターン値として201を通知します。

また、バージョン2. 00以前の場合、リターン値は−1になります。

FPコール番号50～56はバージョン2. 00以降のサポートになりますので、利用の際は必ずこのFPコールでバージョン番号を確認してください。

FPコール番号：51

機　　能　日本語FPが使用しているメイン辞書のバージョンを通知します。

引　　数　なし

コ ー ル

```
MOVE. L     #51, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　バージョン番号またはエラーコード

＝ 0……エラー

＞ 0……バージョン番号

解　　説　日本語FPが現在使用している、メイン辞書のバージョン番号を通知します。
バージョンが2.01の場合には、リターン値として201を通知します。
なお、このFPコールは、日本語FPのバージョン2.00以降よりサポートされています。

FPコール番号：52

機能	環境ファイルを読み込み日本語FPを再設定します。
引数	ENVNAMA　環境ファイル名の文字列の先頭アドレス

コール

```
PEA       ENVNAME
MOVE. L   #52, -(SP)
DC. W     _KNJCTRL
ADDQ. L   #8, SP
```

リターン

0またはエラーコード
　＝ 0………正常終了
　≠ 0……エラー

解説

指定された環境ファイルを読み込み、内容にしたがって日本語FPの環境を再設定します。
なお、このFPコールは、日本語FPのバージョン2.00以降よりサポートされています。

FPコール番号：53

機　　能　漢字変換を含めたリアルタイムキー入力

引　　数　なし

コ ー ル
```
MOVE. L     #53, −(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　0または文字コード
　＝ 0………キー入力なし
　≠ 0……入力文字コード

解　　説　DOS コール inpout（$FF06）の CODE=$FF に相当し、キー入力がない場合でも入力待ちを行いません。
キーが押されていないときには0を通知し、キーが押されていたときには、その文字コードが漢字（全角文字）の場合でも1バイトずつ通知します。
DOS コールをしないで漢字入力をしたいときに、この FP コールを使います。
なお、この FP コールは、日本語 FP のバージョン 2.00 以降よりサポートされています。

FPコール番号：54

機　　能	エコーモード設定

引　　数

MODE　　エコーモード／システムラインモード
　　= 0……システムラインモード
　　= 1……エコーモード

コ ー ル

```
MOVE. L    MODE, -(SP)
MOVE. L    #54, -(SP)
DC. W      _KNJCTRL
ADDQ. L    #8, SP
```

リターン

0またはエラーコード
　= 0………正常終了
　≠ 0……エラー

解　　説

かな漢字変換の表示を画面の最下行で行う場合はシステムラインモードを、画面のカーソル位置で行う場合にはエコーモードを設定します。
設定されているモードを知るときは、FPコール番号55を使用してください。
なお、このFPコールは、日本語FPのバージョン2. 00以降よりサポートされています。

FPコール番号：55

機　　能　エコーモードの通知

引　　数　なし

コ ー ル

```
MOVE. L     #55, -(SP)
DC. W       _KNJCTRL
ADDQ. L     #4, SP
```

リターン　エコーモード

　= 0……システムラインモード

　= 1……エコーモード

解　　説　かな漢字変換の表示を画面の最下行で行っている場合はシステムモードを、カーソル行で行っている場合はエコーモードを通知します。

モード設定方法はFPコール番号54をご覧ください。

なお、このFPコールは、日本語FPのバージョン2.00以降よりサポートされています。

FPコール番号：56

機　能　エコーモードのロック／アンロックの設定

引　数　MODE　ロック／アンロック指定
＝ 0……ロック
＝ 1……アンロック

コール

```
MOVE. L   MODE, －(SP)
MOVE. L   #56, －(SP)
DC. W     _KNJCTRL
ADDQ. L   #8, SP
```

リターン　0またはエラーコード
＝ 0……正常終了
＝ 2……モード指定の誤り

解　説　キーボードから、エコー／システムラインモードの切り替えができないようにロックするか、切り替えられるようにアンロックにするかを設定します。
なお、このFPコールは、日本語FPのバージョン2.00以降よりサポートされています。

第 5 章

BASIC 外部関数の作成

外部関数とは
BASIC 外部関数の仕様
BASIC 外部関数作成例

5.1 外部関数とは

X-BASICの特徴の一つに、プログラム内で種々の関数定義ができることがあげられます。
比較的大きなプログラムを作成するときに、内部を小さな機能単位に分け、それぞれを独立したモジュールとして、組み合せることができるのです。
よく考えて作られたモジュールは、その後また別のプログラムを作るときに再利用できるでしょう。
ちょっとした機能変更も、他のモジュールに影響することなく行えます。
関数定義ができることによって、プログラム開発の能率がアップし、同時にデバッグも簡単になります。
BASICのインタプリタにあらかじめ組み込まれた関数群（BASICマニュアルに説明されているステートメントや標準関数）や、プログラム内でモジュールとして定義された関数は、いわば「内部関数」ともいえます。
これに対して、X-BASICでは、BASICインタプリタの外でも関数を定義できます。
これが本章で解説する「外部関数」です。
もともとX-BASIC本体は、ADPCM、グラフィック、FM音源、スプライト、ジョイスティック、マウスなどのハードウエアに関係の深い機能を、本来のインタプリタに含んでいません。
これらの、ハードウエアに依存する機能は、外部関数として切り離して持っています。
こうした外部関数をBASICの起動時に自動的に組み込んでいるのです。
どの関数を組み込むかということは、

BASIC. CNF

というファイルに書き込んでおきます。
このファイルはちょうど、Human68kのCONFIG. SYSにあたり、これから起動するBASICがどれだけのメモリを使い、画面の横幅をどれだけにし、エラー時のビープ音やキーボードのCAPSスイッチなどの状態をどうするか、などを決めるファイルです。
この中に“FUNC=”というエントリがあり、ここに外部関数ファイルの名前を書いておくのです。

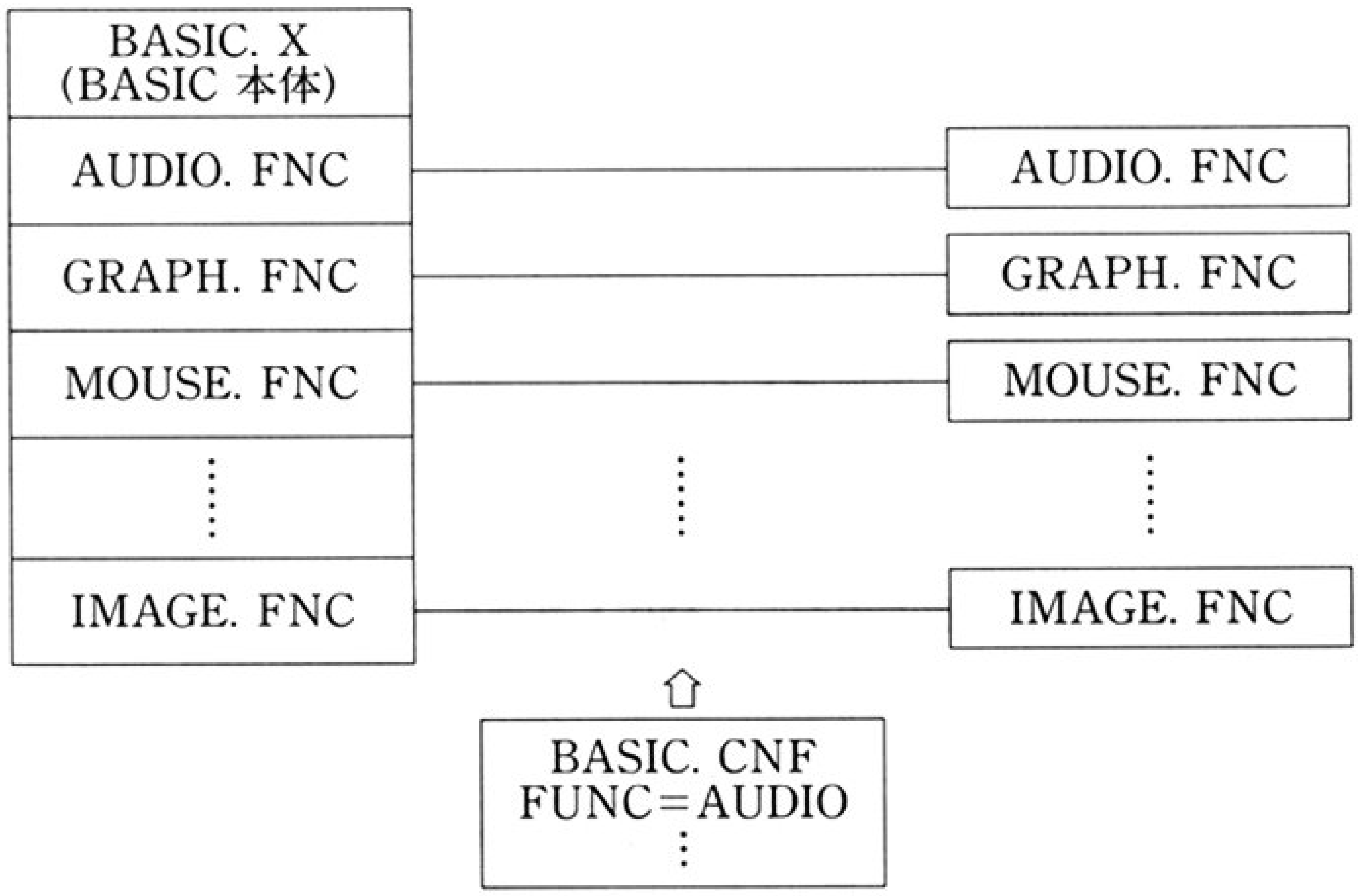
ちなみに、システムディスクの¥BASIC2ディレクトリにあるBASIC.CNFファイルの内容は、次のようになっています。

```
FREE    = 128
WIDTH   = 64
BEEP    = ON
CAPS    = OFF
FUNC    = AUDIO
FUNC    = GRAPH
FUNC    = MOUSE
FUNC    = MUSIC2
FUNC    = SPRITE
FUNC    = STICK
FUNC    = IMAGE
```

デフォルトで"FUNC="以降に書かれている外部関数ファイルの名前は、X-BASICが標準で供給している外部関数です。
それぞれ次のような機能を持っています。

ADPCMの制御	AUDIO. FNC
グラフィックの制御	GRAPH. FNC
マウスの制御	MOUSE. FNC
ミュージックの制御	MUSIC2. FNC
スプライトの制御	SPRITE. FNC
ジョイスティック入力	STICK. FNC
イメージ関係の制御	IMAGE. FNC

もちろん、BASIC. CNFから外してしまった外部関数は、BASIC起動時にインタプリタ内に組み込まれませんから、このBASICではそれらの機能を使えなくなります。
これを利用して、大きな配列をとるなど、どうしても広いメモリ領域をBASICで使用したい場合などは、外部関数を外しておくということも可能です。


BASIC. X
(BASIC 本体)
AUDIO. FNC
GRAPH. FNC
MOUSE. FNC
IMAGE. FNC
AUDIO. FNC
GRAPH. FNC
MOUSE. FNC
IMAGE. FNC
BASIC. CNF
FUNC=AUDIO

5.2 BASIC 外部関数の仕様

ここで解説する「外部関数の仕様」に基づいてさえいれば、ユーザーが自由に任意の外部関数を作成、追加できます。
関数名、入力パラメータの数は自由に設定できます（ただし、戻り値は一つだけ）。
もちろん機械語の外部関数ですから、動作も高速です。
外部関数は、決められたフォーマットにしたがって、アセンブラや各種のコンパイラで作成します。
コンパイル、アセンブル、リンクと手順は、普通のプログラム作成と変わりありません。
アセンブル・リンク後は実行可能な"＊. X"形式のファイル（リロケータブルバイナリファイル）ですが、それを"＊. FNC"という外部関数特有のファイル名にするだけです。
X-BASICを拡張する手順は次の通りです。

①外部関数をまとめた外部関数ファイルを作成します。
- C言語やアセンブリ言語などで、外部関数の機能単位モジュールを作成します。
- アセンブラで「インフォメーションテーブル」（後述）を作成し、外部関数の機能単位モジュールをまとめてインフォメーションテーブルの後につけます。

②上記①で作成した外部関数ファイルのファイル名の拡張子を、". FNC"に変更します。

③EDなどのエディタを起動して、作成した外部関数ファイルの名前をBASIC. CNFに
FUNC＝主ファイル名（拡張子は省略）
と書き込みます。

④BASIC. X、BASIC. CNF、外部関数ファイル（. FNC）を同一ディレクトリに格納し、X-BASICを起動します。

このように、"＊. FNC"ファイルには、複数個の外部関数を入れることが可能です。
たとえば、前述の標準添付の"AUDIO. FNC"には、"A_PLAY"や"A_REC"などの複数の外部関数が、"GRAPH. FNC"には19個もの外部関数が含ま

れています。
これは、外部関数1つについて、1つずつファイルが必要になっては繁雑だという理由からです。
そこで、1つのファイルに複数個の外部関数を書けるよう、各FNCファイルの先頭に「インフォメーションテーブル」という名前のヘッダをつけなければなりません。

5.2.1 インフォメーションテーブル

外部関数ファイルの先頭には、その外部関数ファイルに含まれる関数についての情報を並べた「インフォメーションテーブル」があります。
X-BASICのインタプリタは、外部関数ファイルのインフォメーションテーブルの内容を見て、呼び出された外部関数がどの外部関数ファイルに含まれているのか、その関数のパラメータはいくつか、パラメータの型はなにか、各外部関数の本体の実行アドレスはどこかということがわかるのです。
インフォメーションテーブルは、64バイトのサイズがあります。
構造は以下のようになっており、最初の44バイトはさまざまなルーチンやテーブルのアドレスが入っています。
後の20バイトは予備として、0を入れておいてください。

アドレスのオフセット値	サイズ(バイト)	内容
+0	4	BASIC起動時の初期化ルーチンのアドレス
+4	4	RUN時に実行されるサブルーチンのアドレス
+8	4	END時に実行されるサブルーチンのアドレス
+12	4	SYSTEMコマンド実行時やEXIT命令によるOS復帰時に実行されるサブルーチンのアドレス
+16	4	[BREAK]や、[CTRL]+[C]によるプログラムの中断時に呼ばれるサブルーチンのアドレス
+20	4	一行入力中の[CTRL]+[D]の押下時に呼ばれるサブルーチンのアドレス
+24	4	(予備)
+28	4	(予備)
+32	4	トークンテーブルの先頭アドレス
+36	4	パラメータテーブルの先頭アドレス

+40	4	実行アドレステーブルの先頭アドレス
+44～63	20	(予備) 0を入れておく

注：各テーブルは偶数アドレスになければなりません。

0～23バイト目までは、BASIC上で外部関数を使っているときに起きる可能性のある重大な操作 (BREAKキー (またはCTRL+C) を押すなど) が行われたとき、この外部関数ではどのような処置をとるのかを書いておくアドレスです。
つまり、そうしたことが起きると、起きたことに応じてこのテーブルが指す番地から書かれたプログラムが実行を始めるわけです。

● BASIC 起動時の初期化ルーチンのアドレス
(インフォメーションテーブルの0～3バイト目まで)
X-BASICを起動したときに、この外部関数がどのような処理をするかを記述したプログラムの実行番地を指します。
もし、ここが指すアドレスにRTS命令が書かれていれば、BASIC起動時には特別なことは何もしません。

● RUN 時に実行されるアドレス (4～7バイト目まで)
RUN命令がX-BASICインタプリタ上で実行されたとき、この外部関数がどのような動作をするかを記述したプログラムのアドレスを指しています。
もし、ここが指すアドレスにRTS命令が書かれていれば、RUN命令実行時には特別なことは何もしません。

● END 時に実行されるアドレス (8～11バイト目まで)
END命令がX-BASICインタプリタ上で実行されたとき、この外部関数がどのような動作をするかを記述したプログラムのアドレスを指しています。
もし、ここが指すアドレスにRTS命令が書かれていれば、END命令実行時には特別なことは何もしません。

● SYSTEM／EXIT 時に実行されるアドレス (12～15バイト目まで)
X-BASICインタプリタでは、ダイレクトモードでSYSTEM命令が実行されると、X-BASICを抜けて親プロセスに戻ります。
このような場合に、この外部関数がどのような動作をするかを記述したプログラムのアドレスを指しています。
もし、ここが指すアドレスにRTS命令が書かれていれば、SYSTEM／

EXIT 命令実行時には特別なことは何もしません。

●BREAKまたはCTRL+Cが押されたときに実行されるアドレス
(16〜19 バイト目まで)

X-BASIC インタプリタが外部関数を実行中に、BREAKまたはCTRL+Cキーが押されたとき（ブレークされたとき)、どのような動作をするかを記述したプログラムのアドレスを指しています。
もし、ここが指すアドレスに RTS 命令が書かれていれば、ブレーク時には特別なことは何もしません。

● X-BASIC の一行入力中にCTRL+Dが押されたときに実行されるアドレス (20〜23 バイト目まで)

X-BASIC インタプリタがコマンド待ちのときにCTRL+Dキーが押されたら、どのような動作をするかを記述したプログラムのアドレスを指しています。
もし、ここが指すアドレスに RTS 命令が書かれていれば、一行入力時にCTRL+Dキーが押されても特別なことは何もしません。
X-BASIC では、コマンド待ち時にCTRL+Dが押されると、外部関数をイニシャライズします。
したがって、CTRL+Dで関数をリセットしたいときなどに、ここにリセットルーチンのアドレスを書いておきます。

●（予備＝RTS 命令） (24〜31 バイト目まで)

この後は（予備）のアドレスが2ロングワード続きます。
ここは、将来の拡張に備えた予備用の領域になっています。
将来の互換性を考え、RTS 命令が入ったアドレスを指しておいてください。

●トークンテーブルの先頭アドレス (32〜35 バイト目まで)

定義した外部関数の名前を並べたテーブルのアドレスを指しています。
トークンとは外部関数の名前のことで、X-BASIC インタプリタが外部関数を呼び出すときには、ここに書かれた名前を使います。
トークンテーブルには、この外部関数ファイルで定義する外部関数のすべての名前が格納されています。
関数名は、64 文字までの英数字でなければなりません。
トークンテーブルの詳細については、「5. 2. 2 トークンテーブルの構造」を参照してください。

●パラメータテーブルの先頭アドレス (36～39バイト目まで)

個々の外部関数のパラメータに関するテーブルのアドレスを指しています。

外部関数は当然ながらパラメータつきで呼び出すことができます。

個々の関数がパラメータ（引数）を必要としているかどうか、必要ならその型と数、およびパラメータの順序などの情報を記述したテーブルを「パラメータテーブル」と呼んでいます。

X-BASICインタプリタが外部関数を呼び出すときには、このテーブルが示す場所の情報を使います（ただしパラメータは10個まで)。

注意しなければならないのは、ここに書かれているアドレスが示す先には、各外部関数のパラメータ情報が、直接書かれているわけではないということです。

ここにはパラメータテーブルの先頭アドレスがあり、パラメータテーブルには各外部関数のパラメータに関する情報が書かれた「パラメータIDテーブル」を指すアドレスが書かれています。

各外部関数のパラメータに関する情報は、このパラメータIDテーブルに書かれています。

パラメータテーブルの詳細については、「5. 2. 3 パラメータテーブルの構造」を参照してください。

●実行アドレステーブルの先頭アドレス (40～43バイト目まで)

各トークン（関数名）ごとの、実行アドレスを指しています。

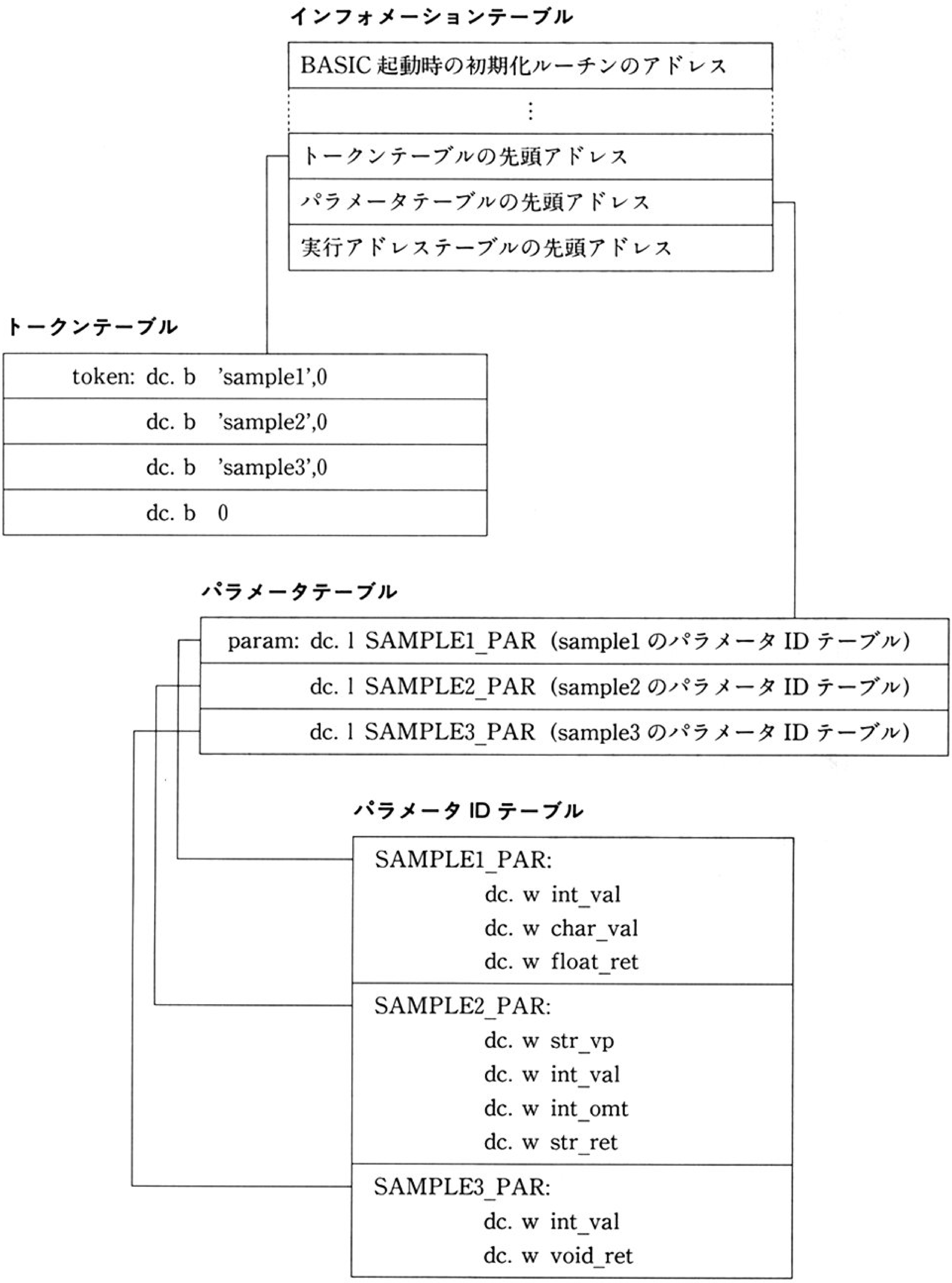
実行アドレスは、トークンテーブルと同じ順序に並んでいます。

このアドレスから先が実際の外部関数処理ルーチンです。

このように、インフォメーションテーブルはかなり複雑な形をしています。

しかし、個々の外部関数が1つ1つファイルになっているよりも、まとめてこのようなテーブルで管理するほうが簡単です。

たとえ1つの外部関数ファイルに1つの外部関数しか含まれていないにしても、このインフォメーションテーブルは必要ですから、ぜひこのテーブルの構造をよく理解し、外部関数を作成してみてください。


インフォメーションテーブル
BASIC 起動時の初期化ルーチンのアドレス
⋮
トークンテーブルの先頭アドレス
パラメータテーブルの先頭アドレス
実行アドレステーブルの先頭アドレス
トークンテーブル
token: dc. b 'sample1',0
dc. b 'sample2',0
dc. b 'sample3',0
dc. b 0
パラメータテーブル
param: dc. l SAMPLE1_PAR (sample1 のパラメータ ID テーブル)
dc. l SAMPLE2_PAR (sample2 のパラメータ ID テーブル)
dc. l SAMPLE3_PAR (sample3 のパラメータ ID テーブル)
パラメータ ID テーブル
SAMPLE1_PAR:
dc. w int_val
dc. w char_val
dc. w float_ret
SAMPLE2_PAR:
dc. w str_vp
dc. w int_val
dc. w int_omt
dc. w str_ret
SAMPLE3_PAR:
dc. w int_val
dc. w void_ret

5.2.2 トークンテーブルの構造

個々の外部関数名は、トークンテーブルに書かれています。
X-BASIC インタプリタは、このテーブルに書かれた関数名で外部関数を呼び出します。
また、この後のパラメータテーブル、実行アドレステーブルもこのトークンテーブルに書かれた順序をもとに並んでいます。
関数名には最大の 64 文字までの英数字が使用できます。関数名の最後はヌル文字（'0'）を書き、これが 1 つのトークンとトークンの区切り（ターミネータ）と見なされます。
このような関数名（トークン）を複数個（1 つでもよい）並べ、トークンテーブルが構成されています。
また、トークンテーブルの最後には、テーブルの終わりを示すエンドコードとして 0 を置きます。
したがって、トークンテーブルの最後には 0 が 2 つ続きます。
また、同じ関数名（トークン）が複数個同じテーブルに定義されている場合は、先に登録された関数名が有効となり、後で定義された関数名は無視されます。

(例) sample 1、sample 2、sample 3 という関数名を登録します。

```
token:  dc. b  'sample1',0
        dc. b  'sample2',0
        dc. b  'sample3',0
        dc. b  0
```

5.2.3 パラメータテーブルの構造

パラメータテーブルは、個々の外部関数のパラメータ（引数）と戻り値に関する情報（パラメータの数（10個まで）、パラメータ、戻り値の型や順序）を管理するテーブルです。
パラメータテーブルには、実際のパラメータに関する情報を書いておくわけではありません。
ここには単に、個々の関数のパラメータIDテーブルのアドレスが書かれています。
そして、パラメータIDテーブルにはじめて、パラメータと戻り値の情報（パラメータIDといいます）が並んでいます。

```
（例） param:  dc.l  SAMPLE1_PAR
               dc.l  SAMPLE2_PAR
               dc.l  SAMPLE3_PAR
```

このように複雑な方法でパラメータを管理しているのは、外部関数によってパラメータの数や大きさ、戻り値の大きさが異なっているためです。
というのは、X-BASICインタプリタが実行しようとする外部関数のパラメータの有無、呼び出されたときのパラメータの型の可否を知るには、インフォメーションテーブルを見るしか方法がありません。
パラメータの数や戻り値の有無は外部関数の設計によって千差万別ですから、これをインフォメーションテーブルにそのまま書いていたのでは、個々の関数ごとにテーブルの大きさが可変になってしまい、次の関数の検索が非常に面倒です。
つまり、上図のパラメータIDテーブルのように、'SAMPLE1_PAR'と'SAMPLE2_PAR'の間の距離と'SAMPLE2_PAR'と'SAMPLE3_PAR'との間の距離が異なってしまいます。
ところが、このような方法ですと、パラメータテーブルの'SAMPLE1_PAR'から'SAMPLE2_PAR'に進むには単に4バイト加えるだけで済みます（ロングワードのアドレスが4バイトであることに注意）。
こうしてパラメータテーブルは、X-BASICインタプリタからのアクセスを高速化しています。

パラメータおよび戻り値は、パラメータIDテーブルに「パラメータID」という形で格納されています。

パラメータIDは長さ1ワードで、次表に示すような数値です（表中は16進で表記）。

パラメータ

パラメータID	意　味	fdef. hによる名前
$0001	8バイト浮動小数点型実数	float_val
$0002	4バイト符号つき整数	int_val
$0004	1バイト符号なし整数	char_val
$0008	文字列	str_val
$0011	浮動小数点型のデータ部のポインタ	float_vp
$0012	int型変数のデータ部のポインタ	int_vp
$0014	char型変数のデータ部のポインタ	char_vp
$0018	文字列型変数のデータ部のポインタ	str_vp
$003F	1次元配列(すべての型)	ary1
$0032	1次元配列(int型)	ary1_i
$0037	1次元配列(浮動小数点型、int型、char型)	ary1_fic
$0034	1次元配列(char型)	ary1_c
$0054	2次元配列(char型)	ary2_c
$0081	省略可能な8バイト浮動小数点型実数	float_omt
$0082	省略可能な4バイト符号つき整数	int_omt
$0084	省略可能な1バイト符号なし整数	char_omt
$0088	省略可能な文字列	str_omt

戻り値

パラメータID	意　味	fdef. hによる名前
$8000	8バイト浮動小数点型実数	float_ret
$8001	4バイト符号つき整数	int_ret
$8003	文字列	str_ret
$FFFF	戻り値なし	void_ret
※ char型は戻り値としては存在しません。		

このように、パラメータIDの値には意味にともなう規則性があります。
まとめると次のようになります。

・ビット15が、　1：戻り値
　　　　　　　　0：パラメータ
・ビット7が、　1：省略可能
　　　　　　　　0：省略不可
・ビット5と6が、D6　D5
　　　　　　　　　0　0　：単純変数
　　　　　　　　　0　1　：1次元配列
　　　　　　　　　1　0　：2次元配列
・ビット4が、　1：ポインタ
　　　　　　　　0：数値
・ビット3が、　1：文字列型
　　　　　　　　0：それ以外
・ビット2が、　1：char型（1バイト符号なし整数）
　　　　　　　　0：それ以外
・ビット1が、　1：int型（4バイト符号つき整数）
　　　　　　　　0：それ以外
・ビット0が、　1：float型（8バイト浮動小数点型実数）
　　　　　　　　0：それ以外

(例)　次のようなパラメータIDテーブルを持つ外部関数は、
　①パラメータは3つ。
　②第1引数はint型数値、第2引数はfloat型数値、第3引数は省略可能で、float型数値。
　③戻り値なし。
ということを表します。

```
param :         dc. w   $0002
                dc. w   $0001
                dc. w   $0081
                dc. w   $FFFF
```

※ "fdef. h" について

パラメータIDは数値ですから、そのままプログラムに使用すると、可読性が悪くなります。

本ツールキットには"fdef. h"というヘッダファイルがあり、あらかじめそれぞれのIDに名前がつけられています。
外部関数プログラムの始めに"fdef. h"をインクルードしておくと誰にもわかりやすく便利です。
前の例を"fdef. h"を使用して書きなおすと、

```
.include    fdef.h
              :
param:      dc.w    int_val
            dc.w    float_val
            dc.w    float_omt
            dc.w    void_ret
```

となります。

5.2.4 パラメータの受け渡し

(1) パラメータの受け渡し

ここでは、実際にX-BASICインタプリタから外部関数が呼ばれたときのパラメータの受け渡し方法について説明します。
X-BASICインタプリタと外部関数との間では、パラメータはスタック経由で受け渡されます。
次の表はその状態を示します。
ちなみに「オフセット」は、外部関数が呼び出される直前のレジスタA7の値（スタックポインタのアドレス）からのオフセットを表します。

オフセット	サイズ(バイト)	内　　容
4	2	パラメータの総個数
6	10	パラメータ1
16	10	パラメータ2
26	10	パラメータ3
⋮	⋮	⋮
(n−1)×10+6	10	パラメータn

なお、パラメータテーブルで指定した以上のパラメータは無効です。
また、パラメータは最大 10 個までです。

(2) パラメータのフォーマット

スタックを介して引き渡されるパラメータは各 10 バイトありますが、その内訳は次の通りです。

オフセット	サイズ(バイト)	内　　容
0	2	パラメータの型
2	4	値の上位 4 バイト
6	4	値の下位 4 バイト

●パラメータの型

X-BASIC インタプリタから外部関数に渡されるパラメータは、その値を見ただけではどのような種類のものか区別できません。
そこで、X-BASIC インタプリタは、外部関数を呼び出してパラメータをスタックにプッシュするさいに、そのパラメータの型を値とともに伝えます。
このことは、型によって意味のある値のバイト数が異なることを考えれば重要です。

型を示す値	パラメータの型
0	8 バイトの浮動小数点型実数（float 型）
1	4 バイトの符号つき整数（int 型）
2	1 バイトの符号なし整数（char 型）
3	文字列型（str 型）
$FFFF	省略された引数（void 型）

X-BASIC インタプリタは、パラメータを外部関数に渡すために、パラメータ 1 つあたり 10 バイトのスタック領域を使います。
このうち最初の 2 バイトは、渡すパラメータの型を示す上の表の値です。
残りの 8 バイトは（原則として）そのパラメータの実際の値です。
この 8 バイトを FAC（Floating ACcumulator）といいます。
しかし、型によってパラメータの実効バイト数が異なります。

すなわち、次のようになります。

・float 型の場合は、8バイトすべてを使用します。
・int 型の場合には、8バイトのうち下位4バイトを使用します。
　上位4バイトにはすべて0が入ります。
・char 型の場合には、8バイトのうち下位1バイトだけを使用し、上位7バイトにはすべて0が入ります。
・文字列型の場合には、8バイトのうち下位4バイトだけを使用し、上位4バイトにはすべて0が入ります。
　下位4バイトには文字列そのものが入るのではなく、文字列型のデータ部へのポインタとして、メモリアドレスが入ります。
・ポインタの場合には、文字列型と同じく、8バイトのうち下位4バイトだけを使用し、上位4バイトにはすべて0が入ります。
　下位4バイトには、データ部へのポインタとしてメモリアドレスが入ります。
・配列の場合には、文字列型と同じく、8バイトのうち下位4バイトだけを使用し、上位4バイトにはすべて0が入ります。
　下位4バイトには、配列へのポインタとしてメモリアドレスが入ります。

型の指定を除く8バイトに、どのように値が格納されているかを知るためにも、最初の2バイトの型の指定は重要なのです。
同様に、どんな値がX-BASICインタプリタから渡されるかわからないような場合にも、この型の指定を見れば判断できます。

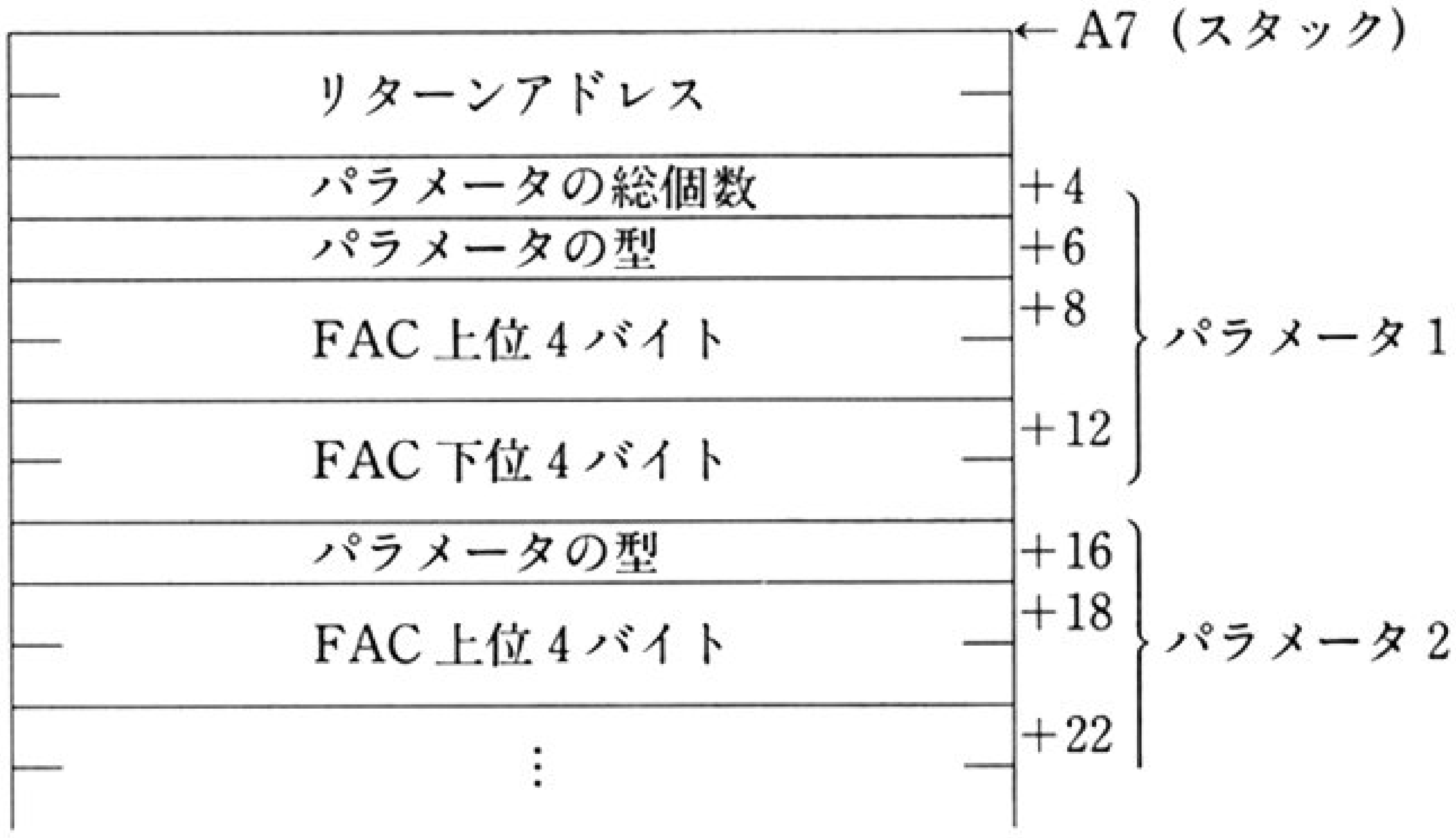
＜FACの構造＞

	上位4バイト				下位4バイト			
float 型	データ							
int 型	0	0	0	0	データ			
char 型	0	0	0	0	0	0	0	データ
文字列型	0	0	0	0	ポインタ			
ポインタ	0	0	0	0	ポインタ			

配列	0	0	0	0	ポ イ ン タ

※ FAC……Floating ACcumulator：引数などの値を格納する場所のこと。

〈パラメータの構造〉



〈配列ポインタの構造〉

オフセット	サイズ（バイト）	内　　容
0	4	スキップオフセット
4	2	次元数−1
6	2	データのサイズ(1, 4, 8)
8	2	添字の大きさ(0〜65535)
(10)	(2)	(2 次元目の添字の大きさ)
(12)	(4)	(2 次元の場合(?, 0)の先頭のオフセット)
10(16)	各データのサイズ	データ領域

※カッコ内は 2 次元配列の場合

(例)　1 次元配列の場合のデータの引渡し

```
par1   equ       6
       move. l   par1+6(sp), a1
       move. w   6(a1), d1        *データのサイズ(1, 4, 8)
       move. w   8(a1), d0        *添字の大きさ(0~65535)
       addq. w   #1, d0           *0でも1個あり
       mulu. w   d1, d0           *データ部の総数
       lea       10(a1), a1       *データ部の先頭ポインタ
```

＊ a1 から d1 バイト単位で総数 d0 バイトが配列のデータ部

例として、"S_ASCII" という名前の外部関数を、

S_ASCII (sharp, x68, tech)

というパラメータで呼んだとし、スタックを見てみましょう。
ただし "sharp" は浮動小数点型数値、"x68" は文字列型データ部へのポインタ、"tech" は省略可能な char 型数値とし、戻り値は符号つき整数です。
このような場合、呼び出された直後のスタックの状態は次のようになっているはずです。

スタックの内容	オフセット
(外部関数実行完了後に戻るリターンアドレス)	← A7 (スタック)
3 (パラメータの総個数)	+4
0 (1 番目のパラメータ "sharp" の型)	+6
("sharp" の値の上位 4 バイト)	+8
("sharp" の値の下位 4 バイト)	+12
3 (2 番目のパラメータ "x68" の型)	+16
0 ("x68" が文字列型なので、ここは使用しない)	+18
("x68" のデータ部へのポインタ)	+22
2 (3 番目のパラメータ "tech" の型)	+26
0("tech" が char 型なので、ここは使われない)	+28
("tech" の値)	+35

5.2.5 戻り値の受け渡し

パラメータとは異なり、外部関数からX-BASICインタプリタへの戻り値は1つ(または0)しかありません。
そこで、戻り値はレジスタを介して引き渡されます。

D0.L ……………　エラーコード（0はエラーなし）
A0.L ……………　戻り値を格納したエリアへのポインタ
A1.L ……………　エラーメッセージへのポインタ（エラー時のみ有効）

外部関数は、その実行が正常に終了したかどうかをX-BASICインタプリタに知らせるのに、D0レジスタを使います。
外部関数の処理結果がエラーなら、エラーコードをD0.Lに入れ、表示したいエラーメッセージを格納したエリアのアドレスをA1.Lに入れてリターンします。
X-BASICインタプリタは、D0.Lが0でなかったら指定されたエラーメッセージを表示してプログラムの実行を中断します。
エラーでなければ(正常終了なら)、D0.Lを0にし、戻り値を格納したエリアのアドレスをA0.Lに入れてリターンします。
戻り値を格納するエリアは10バイト必要です。
格納の方法は、外部関数にパラメータを引き渡すときのスタックの状態に似ています。
次のようにしてください。

オフセット	サイズ（バイト）	内　容
0	2	ワードで0を入れておく
2	4	戻り値の上位ロングワード
6	4	戻り値の下位ロングワード

値を返さない外部関数を実行した場合は、X-BASICインタプリタはA0レジスタを無視しますから、何が入っていても構いません(エラーだった場合も同様)。

5.3 BASIC外部関数作成例

ここに掲げたサンプルプログラム"usfunc. s"は、次の3つの外部関数を含んでいます。

1. Human68k上で設定されている各種の環境変数とその値を得る"GETENV"
   環境文字列はHuman68kの「SETコマンド」で設定／表示できますが、GETENVはその表示をX-BASIC上から行う関数です。
   環境変数名を引数にして呼び出します。
   環境文字列についての詳細は、X68000本体に添付の「Human68kユーザーズマニュアル」の「SETコマンド」の項を参照してください。
   環境文字列の設定はできません。

2. ディスクの指定位置からデータを直接読み込む"DSKI"
   DSKI関数は、ドライブ、トラック、サイド、セクタ、そして読み込み先のバッファのアドレスを指定して、ディスクに書かれたデータをバッファに直接読み込みます。
   バッファの確保、バッファからのデータの読み出しはユーザーが行ってください。

3. ディスクの指定位置に直接データを書き込む"DSKO"
   DSKO関数は、ドライブ、トラック、サイド、セクタ、そして書き出すデータがあるバッファのアドレスを指定して、バッファのデータをディスクに直接書き込みます。
   バッファの確保、バッファへのデータの書き込みはユーザーが行ってください。

注：DSKO関数は、ディスクを壊す可能性のある危険な関数です。
　　書き込む位置やデータの内容を誤るとディスクを破壊し、復旧は非常に難しくなります。
　　ディスクの内部構造が理解できないうちは、使用しないでください。

5.3　BASIC外部関数作成例

以下に、次のようなプログラムリストを掲載します。

usfunc. s	BASIC外部関数（getenv、dski、dsko）のアセンブリ言語によるソースプログラム
mkfunc. bat	"usfunc. s"から"usfunc. fnc"を作成する手順をまとめたバッチプログラム
testenv. bat	"testenv. bas"でテストする前に環境変数を設定するバッチプログラム
testenv. bas	getenvをテストするためのX-BASICプログラム
disk. bas	dski、dskoをテストするためのX-BASICプログラム

★ usfunc. s

```
*****************************************
* User Function for X-BASIC
*****************************************

        .include        doscall.mac
        .include        iocscall.mac
        .include        fdef.h

iocs    equ     $0f

*-------------------------------
* インフォメーション テーブル
*-------------------------------
        dc.l    X_INIT
        dc.l    X_RUN
        dc.l    X_END
        dc.l    X_SYS
        dc.l    X_BRK
        dc.l    X_CTRL_D
        dc.l    X_RES1
        dc.l    X_RES2
        dc.l    PTR_TOKEN
        dc.l    PTR_PARAM
        dc.l    PTR_EXEC
        dc.l    0,0,0,0,0

X_INIT:
X_RUN:
X_END:
X_SYS:
X_BRK:
X_CTRL_D:
X_RES1:
X_RES2:
        rts

*-------------------------------
* Function Name Table
*-------------------------------
PTR_TOKEN:
        dc.b    'getenv',0
        dc.b    'dski',0
        dc.b    'dsko',0
        dc.b    0

*-------------------------------
* Parameter Table
*-------------------------------
        .even
PTR_PARAM:
        dc.l    GETENV_PAR
        dc.l    DSKINP_PAR
        dc.l    DSKOUT_PAR
```

```
*-------------------------------
* パラメータIDテーブル
*-------------------------------
GETENV_PAR:
        dc.w    str_val         * 引数は文字列型
        dc.w    str_ret         * 戻り値は文字列型

DSKINP_PAR:
        dc.w    int_val         * ドライブ (0～3)
        dc.w    int_val         * トラック (0～76)
        dc.w    int_val         * サイド (0～1)
        dc.w    int_val         * セクタ (0～7)
        dc.w    ary1_c          * ディスクバッファ char buf(1023)
        dc.w    int_ret

DSKOUT_PAR:
        dc.w    int_val         * ドライブ (0～3)
        dc.w    int_val         * トラック (0～76)
        dc.w    int_val         * サイド (0～1)
        dc.w    int_val         * セクタ (0～7)
        dc.w    ary1_c          * ディスクバッファ char buf(1023)
        dc.w    int_ret


*-------------------------------
* Function Address Table
*-------------------------------
PTR_EXEC:
        dc.l    GET_ENVIRON
        dc.l    DISK_INPUT
        dc.l    DISK_OUTPUT

*-------------------------------
* Get Environ
* in    12(sp).l *ENV_NAME
* ret   ENV_STR
*       NULL    (Error)
*-------------------------------
        .even
        .text
GET_ENVIRON:
        move.w  #0,ENV_STR      * 環境変数の内容に NULL をセット
        move.l  par1+6(sp),ENVNAMP * 環境変数名のアドレスをコピー
        pea     ENV_STR         * 書き出しバッファ先頭アドレスをプッシュ

        clr.l   -(sp)           * 環境ポインタ戻り値用に 0 をプッシュ
        move.l  ENVNAMP,-(sp)   * 環境変数名の先頭アドレスをプッシュ
        dc.w    _GETENV         * ファンクションコール
        lea     12(sp),sp       * スタック調整
        cmp.l   #0,d0           * 負数ならエラーへ
        bcs     NO_ENV
SUCCESS:                        *--- 正常終了 ---
        clr.w   RET_DAT         * 戻り値エリアの先頭に0(wordで)を入れる
        lea     ENV_STR,a0      *
        move.l  a0,INT_DAT      * INT_DAT<-- 環境変数の先頭アドレス
        lea     RET_DAT,a0      * a0 に戻り値ブロックの先頭アドレス
        moveq.l #0,d0           * d0 に 0 をセット(正常終了)
        rts
```

```
NO_ENV:                                 *--- 環境変数がない ---
        clr.w   RET_DAT                 * 戻り値エリアの先頭に0(wordで)を入れる
        lea     ENV_STR,a0
        move.l  a0,INT_DAT              * INT_DAT <-- 0
        lea     RET_DAT,a0              * a0 に戻り値ブロックの先頭アドレス
        moveq.l #0,d0                   * d0 に 0 をセット(正常終了)
        rts

*------------------------------
* 戻り値データブロック
*------------------------------
        .data
RET_DAT dc.w    0                       * 戻り値のバッファ
        dc.l    0                       * 上位ロングワード
INT_DAT dc.l    0                       * 下位ロングワード


*------------------------------
* ワークエリア
*------------------------------
ENVNAMP dc.l    0                       * 環境変数名文字列へのポインタ
ENV_STR ds.b    80                      * 環境変数の内容が入るバッファ


*---------------------------------------------
* DISK_INPUT
*       dski(drv, trc, sid, sct, buf)
* in    15(sp).b drv    Drive No.
*       25(sp).b trc    Track
*       35(sp).b sid    Side
*       45(sp).b sct    Sector
*       52(sp).l buf    Buffer Top Address char buf(1023)
*
* ret    0:
*       -1: IOCS Call Error
*---------------------------------------------
        .even
        .text
DISK_INPUT:                             * パラメータをワークにコピー
        move.b  par1+9(sp),DRIVE
        move.b  par2+9(sp),TRACK
        move.b  par3+9(sp),SIDE
        move.b  par4+9(sp),SECTOR
        move.l  par5+6(sp),BUFADR
        bsr     DISK_PARASET            * IOCS のパラメータの形式にします
        moveq.l #_B_READ,d0             * IOCSコール B_READ
        move.w  PDA,d1                  *       PDA
        move.l  HEADPOS,d2              *       リード位置
        move.l  #1024,d3                *       バイト数
        movea.l BUFADR,a1               *       バッファ先頭アドレス
        lea     10(a1),a1
        trap    #iocs                   *       IOCSコール
        btst.l  #30,d0                  * エラー?
        bne     DIO_ERR                 * エラーなら DIO_ERR
        moveq.l #0,d0                   * d0.l <-- 0 (エラーなし)
        lea     DISK_RET,a0             * a0.l <-- 戻り値エリアの先頭アドレス
        clr.w   (a0)                    * 戻り値エリアの先頭に0(wordで)を入れる
        clr.l   6(a0)                   * INT 型の戻り値 0 をセット
        rts                             * BASIC に戻ります
```

```
*-------------------------------------------
* DISK_OUTPUT
*
* dsko(drv, trc, sid, sct, buf)
* in    15(sp).b drv    Drive No.
*       25(sp).b trc    Track
*       35(sp).b sid    Side
*       45(sp).b sct    Sector
*       52(sp).l buf    Buffer Top Address char buf(1023)
*
* ret   0
*       -1 IOCS Call Error
*-------------------------------------------
        .even
        .text
DISK_OUTPUT:
        move.b  par1+9(sp),DRIVE
        move.b  par2+9(sp),TRACK
        move.b  par3+9(sp),SIDE
        move.b  par4+9(sp),SECTOR
        move.l  par5+6(sp),BUFADR
        bsr     DISK_PARASET      * IOCS のパラメータの形式にします
        moveq.l #_B_WRITE,d0      * IOCSコール B_WRITE
        move.w  PDA,d1            *       PDA
        move.l  HEADPOS,d2        *       ライト位置
        move.l  #1024,d3          *       バイト数
        movea.l BUFADR,a1         *       バッファ先頭アドレス
        lea     10(a1),a1
        trap    #iocs             *       IOCSコール
        btst.l  #30,d0            * エラー?
        bne     DIO_ERR           * エラーなら DIO_ERR
        moveq.l #0,d0             * d0.l <-- 0 (エラーなし)
        lea     DISK_RET,a0       * a0.l <-- 戻り値エリアの先頭アドレス
        clr.w   (a0)              * 戻り値エリアの先頭に0(wordで)を入れる
        clr.l   6(a0)             * INT 型の戻り値 0 をセット
        rts                       * BASIC に戻ります

*--- ERROR ROUTINE for DISK_INPUT / DISK_OUTPUT ---*
DIO_ERR:
        moveq.l #0,d0             * エラーで終了
        lea     DISK_RET,a0       * a0.l <-- 戻り値エリアの先頭アドレス
        clr.w   (a0)              * 戻り値エリアの先頭に0(wordで)を入れる
        move.l  #-1,6(a0)         * INT型の戻り値 -1 をセット
        rts

*--- DATA CONV. BASIC-->IOCS ---*
DISK_PARASET:                     *--- PDA ---
        move.b  DRIVE,d0          * d0.b <-- $0x (x 0…2)
        cmp.b   #0,d0             * if (drv < 0)
        bcs     DRV_ERR           *       goto DRV_ERR
        cmp.b   #2,d0             * if (drv > 2)
        bgt     DRV_ERR           *       goto DRV_ERR
        add.b   #$90,d0           * d0.b <-- $9x
        lsl.l   #8,d0             * d0.w <-- $9x00
        move.b  #$50,d0           * d0.w <-- $9x50
        move.w  d0,PDA            * PDA <-- $9x50
                                  *--- Position ---
        moveq.l #3,d0             * d0.l <-- $0000_0003 セクタ長コード
        lsl.l   #8,d0             * d0.l <-- $0000_0300
        move.b  TRACK,d0          * d0.l <-- $0000_03tt $tt:トラック番号
        lsl.l   #8,d0             * d0.l <-- $0003_tt00
        move.b  SIDE,d0           * d0.l <-- $0003_ttss $ss:サイド
        lsl.l   #8,d0             * d0.l <-- $03tt_ss00
        move.b  SECTOR,d0         * d0.l <-- $03tt_ssrr $rr:セクター番号
        move.l  d0,HEADPOS        * HEADPOS <-- $03ttssrr
        rts
```

```
*--- BAD DRIVE No. ---
DRV_ERR:
        moveq.l #1,d0
        lea     DRV_MSG,a1
        addq.l  #4,sp           * スタックレベル調整
        rts

*------------------------------
* WORK AREA for DSKI / DSKO
*------------------------------
        .data
DRIVE   dc.b    0
TRACK   dc.b    0
SIDE    dc.b    0
SECTOR  dc.b    0
BUFADR  dc.l    0

PDA     dc.w    0
HEADPOS dc.l    0

DISK_RET
        dc.w    1
        dc.l    0
        dc.l    0

DRV_MSG dc.b    "Bad Drive No. ",$0d,$0a,0

        .end
```

★ mkfunc. bat

```
as usfunc
lk /ousfunc.fnc usfunc
```

★ testenv. bat

```
set path=c:¥bin;c:¥;a:¥bin;a:¥;
set met=This is test Env.
```

★ testenv. bas

```
100 /* save"test"
110 /*--------------------*/
120 /*- Test Program     -*/
130 /*- UFUNC.FNC        -*/
140 /*-     getenv(str)  -*/
150 /*-                  -*/
160 /*--------------------*/
170 dim str env
180 env = getenv("")
190 print env
200 env = getenv("met ")
210 print env
220 print getenv("path")
230 end
```

★ disk. bas

```
100 /*    save "disk"
110 /*----------------------------*/
120 /*  Sample Prg of Userfunc.    */
130 /*                             */
140 /*----------------------------*/
145   dim char buf(1023)
150   /*--- Main Prg. ---*/
160   dim int drv, trc, sid, sct, ret
170   dim str htrc, hsct
180   width 96
190   drv = 1
200   for trc = 0  to 76
210     for sid = 0 to 1
220       for sct = 6 to 8
230         ret = dski(drv, trc, sid, sct, buf)
240         if ret <> 0 then {
250            print "Error !! "
260            end
270         }
280         htrc = itoh(trc,2)
290         hsct = itoh(sct,2)
300         print
310         print using "Drive #  Track $& & "; drv, htrc;
312         print using "Side #  Sector $& &"; sid, hsct
320         print "--- page 0 ---" : dumpbuf(0)
322         print "--- page 1 ---" : dumpbuf(1)
324         print "--- page 2 ---" : dumpbuf(2)
326         print "--- page 3 ---" : dumpbuf(3)
330       next
340     next
350   next
360   end
370 /*--- Dump Disk Buffer ---*/
380 func int dumpbuf(n)
390   dim int i, c, l, d
400   dim str h
410   dim str s[16]
412   print "  : 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F";
```

```
414   print "    0123456789ABCDEF"
416   print "--------------------------------------------------";
418   print "--------------------"
420   for i = 0 to 255
430     c = i mod 16
440     l = i / 16
450     if c = 0 then {
470         print itoh(i, 2); ":";
480     }
490     d = buf(n * 256 + i)
500     print itoh(d,2); " ";
520     if              d < &H20 then d = '.'
522     if &H7F < d and d < &HA0 then d = '.'
524     if &HDF < d              then d = '.'
530     s[c]=d
540     if c = 15 then print "    "; s
550   next
560 endfunc
570 /*--- itoh ---*/
580 func str itoh(i;int, l;int)
590   str h
600   if l < 1 then {
610     print "Error in itoh
620     end
630   }
640   h = hex$(i)
650   if strlen(h) < l then {
660     h = string$(l, "0") + h
670     h = right$(h, l)
680   }
690   return(h)
700 endfunc
```

第6章
デバイスドライバ

デバイスドライバとは
デバイスドライバのフォーマット
デバイスヘッダの内容
データの受け渡し
キャラクタデバイスドライバ
ブロックデバイスドライバ
BPBテーブル
サンプルプログラム

6.1 デバイスドライバとは

Human68k のシステム内部の構成は、論理的階層ごとにモジュール化されています。

●ユーザーインターフェイス (COMMAND. X)

コンソール (CON) を通じてユーザーと対話し、ユーザーの命令を解釈し、論理層に指令を出します。

●論理層 (HUMAN. SYS)

ファイル、メモリなどの資源を論理的に管理しており、COMMAND. X またはアプリケーションプログラムからの要求を受けて、物理層への指令を出します。

●物理層 (IOCS……ROM)

デバイスを直接操作する階層であり、論理層からの指令に従い、各デバイスをコントロールします。

この層のうち、物理層はデバイスごとに専用のコントロールプログラムが必要です。
このデバイスごとのコントロールプログラムを“デバイスドライバ”と呼びます。
ディスプレイ、キーボード、ディスクなど基本的な入出力装置に対応するデバイスドライバは、HUMAN. SYS にふくまれており、Human68k が起動した際には常に存在しています。
これらを“標準のデバイスドライバ”と呼びます。
Human68k では、新たなデバイスを追加した場合でも、それまでのシステムを変更せずにデバイスドライバを組み込むことができるように考慮されています。
新規のデバイスドライバをシステムに登録する場合には、CONFIG. SYS ファイルの中に、デバイスドライバのファイル名を記述します。
Human68k は、起動時に CONFIG. SYS ファイルを参照し、そこに書かれているデバイスドライバをシステムに組み込みます。
デバイスドライバの組み込みは、CONFIG. SYS ファイルの中に記述してある順で行われます。

つまり、同じデバイス名のドライバが複数あった場合、後のものが有効になります。
これは標準のデバイスドライバにも当てはまります。
このため、Human68k の標準のデバイスドライバ (CON、AUX、PRN など)を新規のデバイスドライバによって置き換えることも可能です。

6.2 デバイスドライバのフォーマット

デバイスドライバには、ファイルストレージエリアのためのブロックデバイスと、外部装置とシリアルな入出力を行うためのキャラクタデバイスの二つの種別があります。
ブロックデバイスはディスクに代表されるもので、ブロック単位で、ランダムな入出力を行います。
またブロックデバイスには固有の名前はつけられず、システムに組み込まれた順に“A：、B：、C：……”という名前が割りあてられます。
キャラクタデバイスはプリンタ(PRN)やRS-232C (AUX)に代表されるもので、文字単位でシリアルな入出力を行います。
キャラクタデバイスは、各デバイスごとに固有の名前を使用し、アプリケーションプログラムからは、その名前でファイルとしてオープンすることによって操作します。
デバイスドライバは、次に示すモジュールから構成されます。

●デバイスヘッダ

デバイスドライバの先頭に位置するブロックであり、そのプログラムがどのようなデバイスドライバであるかを示すと同時に、データ処理ルーチンへのポインタ、およびデバイス名（キャラクタデバイスの場合のみ）を持ちます。

●ストラテジルーチン

このルーチンはHuman68kの要求ごとに呼び出され、処理の内容とデータなどを含む“リクエストヘッダ”と呼ばれるブロックへのポインタを受け取り、一度リターンします。

●割り込みルーチン

ストラテジルーチンが受け取ったリクエストヘッダを実際に処理するルーチンです。
デバイスドライバによる処理のすべてを受け持ちます。
リクエストヘッダ内には、要求する処理の種別がコマンドコードによって示されてあり、各コードに対応した処理ルーチンを用意します。

デバイスドライバの処理ルーチンは、疑似マルチタスク処理のために2つの

モジュールに分けられています。

注：標準入出力デバイス(CON)を作成する場合には、以下の4つのOSファンクションを処理するルーチンを作成したうえ、intvcs($FF25)によって、処理ルーチンを登録しなければなりません。

$FF21　fnckey(MD ＊ 256＋FNO、BUFFER)
$FF22　knjctrl(MD、?)
$FF23　conctrl(MD、?)
$FF24　knjctrl(MD、?)

漢字変換機能を使用するのであれば、メッセージ表示のためにhendsp ($FF18) を使用可能にしておかなければなりません。
また、デバイスドライバ内で辞書を読み書きする場合には、辞書のopen ($FF3D) 時にMODEとして$1?? (辞書専用のモード) を使用してください。
辞書のオープンは同時に2個まで行えます。
なお、break onの設定になっている場合には、CONデバイスは再帰呼び出しが行われることになるので、一時的にbreak offにするか、再帰呼び出しが可能なプログラムにしておく必要があります。

6.3 デバイスヘッダの内容

デバイスドライバは、必ず次のデバイスヘッダで始めなければなりません。

サイズ	内　容
1ロングワード	次のドライバへのリンクポインタ 1つのデバイスドライバの中で2つ以上のデバイスをサポートする場合には、次のデバイスヘッダのアドレスをセットします そのヘッダで終りの場合、および1つしかサポートしない場合－1($FFFFFFFF)にしてください
1ワード	デバイスの層性を表します ビット15　0：ブロックデバイス 　　　　　1：キャラクタデバイス ビット14　0：IOCTRL不可(ioctrl($FF44)使用不可) 　　　　　1：IOCTRL可能(ioctrl($FF44)使用可) ビット12　～　ビット6 　　　　常に0 ビット5　0：COOKED MODE (1バイトずつ処理) 　　　　1：RAW MODE(指定バイトずつ処理) ビット3=1：CLOCKデバイスとします ビット2=1：NULデバイスとします ビット1=1：標準出力デバイスとします ビット0=1：標準入力デバイスとします (ビット5～0はビット15=1の場合のみ有効)
1ロングワード	ストラテジルーチンのエントリポイント リクエストヘッダのポインタはA5レジスタで与えられます

6.3 デバイスヘッダの内容

1ロングワード	割り込みルーチンのエントリポイント
8バイト	デバイス名（キャラクタデバイス） ブロックデバイスの場合、先頭の1バイトは$20以下のコード 残り7バイトには、デバイス名を入れてもかまいません

ここでビット14は、ioctrl($FF44)による入出力を行うデバイスドライバであるかどうかを指定します。
このビットが0であると、ioctrl($FF44)による入出力はできません。
またビット5は、そのキャラクタデバイスがRAWモードであるかCOOKEDモードであるかどうかを指定します。
2つのモードの間ではCTRL+S、CTRL+Zなどのコントロールコードの処理が違います。

6.4 データの受け渡し

デバイスドライバとHuman68kの間では、A5レジスタを使用して、リクエストヘッダへのポインタの形でデータの受け渡しを行います。
リクエストヘッダは処理の内容によって13～23バイトのデータから構成されます。
実際にデータを読み書きするコマンドでは、リクエストヘッダの中にバッファのアドレスがセットされており、このバッファを使用して、データの受け渡しを行います。
デバイスドライバの処理結果（処理完了／エラー発生）は、D0. Wにセットするとともに、リクエストヘッダの中にもセットして戻ります。

●エラーコード

$0000 ： 処理完了（ドライバから抜ける場合、常にセットします）

上位8ビット	=	$10～$1F	エラー（中止のみ）
	=	$20～$2F	エラー（再実行のみ）
	=	$30～$3F	エラー（中止 & 再実行）
	=	$40～$4F	エラー（無視のみ）
	=	$50～$5F	エラー（中止 & 無視）
	=	$60～$6F	エラー（再実行 & 無視）
	=	$70～$7F	エラー（中止 & 再実行 & 無視）
下位8ビット	=	$01	ユニット番号が不正
	=	$02	ドライブの準備ができていない
	=	$03	コマンドコードが不正
	=	$04	CRCエラー
	=	$05	ディスク管理領域が破壊されている
	=	$06	シークエラー
	=	$07	無効なメディア
	=	$08	セクタが見つからない
	=	$09	プリンタオフライン
	=	$0A	書き込みエラー
	=	$0B	読み出しエラー
	=	$0C	その他のエラー
	=	$0D	ライトプロテクト（とり替え不可）
	=	$0E	書き込み不可（とり替え可能）、 またはディスクが入っていない

●コマンドコード

システムがデバイスドライバに要求する処理の内容は、ヘッダ内、2(A5)にあるコマンドコードによって識別します。

コマンドコードには次に示す値が使用されます。

ここで「未使用」は、キャラクタデバイス、ブロックデバイスのどちらも使用していないことを、「エラー」はどちらか一方で使用していることを表します。

キャラクタデバイス

コード	要　求
0	初期化
1	エラー
2	未使用
3	IOCTRLによる入力
4	入力（0以外のデータを待つ）
5	先読み入力（0でも戻る）
6	入力ステータスチェック
7	入力バッファをクリア
8	出力(VERIFY OFF時)
9	出力(VERIFY ON時)
10	出力ステータスチェック
11	未使用
12	IOCTRLによる出力

ブロックデバイス

コード	要　求
0	初期化
1	ディスク交換チェック
2	未使用
3	IOCTRLによる入力
4	入力
5	ドライブコントロール&センス
6	エラー
7	エラー

8	出力(VERIFY OFF 時)
9	出力(VERIFY ON 時)
10	エラー
11	未使用
12	IOCTRL による出力

コマンドコードと要求される処理、リクエストヘッダの形式を次に示します。

なお表中では、

I………デバイスドライバに渡されるデータ
O………デバイスドライバが返すデータ
B………バイト
W………ワード
L………ロングワード

として表記します。

6.5 キャラクタデバイスドライバ

● **初期化**

コマンドコード ＝ 0

リクエストヘッダ

位　置	サイズ	I/O	要　　求
0 (A5)	1B	−	26（定数）
1 (A5)	1B	−	未使用
2 (A5)	1B	I	コマンドコード (0)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)
5 (A5) ⋮ 13 (A5)	9B	−	未使用
14 (A5)	1L	O	デバイスドライバの終りアドレス
18 (A5)	1L	I	パラメータポインタ
22 (A5)	1B	−	未使用

初期化ルーチンは、システム起動時に一回だけ呼ばれます。
この時、パラメータポインタはCONFIG.SYSファイルに記述された“DEVICE＝”の次のデータを指しています。
このポインタを見て、必要なパラメータを受け取り、タイトルを表示し、初期化動作を行った後、デバイスドライバの終りのアドレスをセットして戻ります。

例　“DEVICE＝¥SYS¥PRNDRV. SYS　#/M7”と記述してある場合、
　　以下のparptrを指していることになります。

```
parptr :    dc.b    '¥SYS¥PRNDRV. SYS',0
            dc.b    '#/M7',0
            dc.b    0
```

●入力または出力

コマンドコード　＝　3、4、8、9、12

リクエストヘッダ

位　置	サイズ	I/O	要　求
0 (A5)	1B	−	26（定数）
1 (A5)	1B	−	未使用
2 (A5)	1B	I	コマンドコード（3、4、8、9、12)
3 (A5)	1B	O	エラーコード（low)
4 (A5)	1B	O	エラーコード（high)
5 (A5) ⋮ 13 (A5)	9B	−	未使用
14 (A5)	1L	I	転送バッファのアドレス
18 (A5)	1L	I	バイト数

指定されたバッファへ、指定されたバイト数のデータを読み込みます（入力)。

データが用意できていない場合には、データが揃うまで待ちます。

指定されたバッファから、指定されたバイト数のデータを書き出します（出力)。

コマンドコード9の場合は、書き出した後にそのデータを読み込んで、実際に書き出せていることを確認します。

●先読み入力

コマンドコード　＝　5

リクエストヘッダ

位　置	サイズ	I/O	要　求
0 (A5)	1B	−	26（定数)
1 (A5)	1B	−	未使用
2 (A5)	1B	I	コマンドコード(5)
3 (A5)	1B	O	エラーコード(low)
4 (A5)	1B	O	エラーコード(high)

5 (A5) ⋮ 12 (A5)	8B	−	未使用
13 (A5)	1B	O	データバッファ

デバイスからデータを1バイト先読みし（そのデータを、次に読み込みを行った際に再び読み込めるように残しておく）、バッファにセットして戻ります。

データがない場合には、バッファに$00をセットしてすぐに戻ります。

●ステータスチェック

コマンドコード ＝ 6、10

リクエストヘッダ

位　置	サイズ	I/O	要　　求
0 (A5)	1B	−	26 (定数)
1 (A5)	1B	−	未使用
2 (A5)	1B	I	コマンドコード (6、10)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)

デバイスが入出力可能か否かをチェックし、可能であればステータスとし$00をセットし、不可能であればステータスとして$01をセットして戻ります。

●入力バッファをクリア

コマンドコード ＝ 7

リクエストヘッダ

位　置	サイズ	I／O	要　　求
0 (A5)	1B	－	26 (定数)
1 (A5)	1B	－	未使用
2 (A5)	1B	I	コマンドコード (7)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)

それまでに入力されて、バッファにたまっていたデータをすべて捨てます。

6.6 ブロックデバイスドライバ

●初期化

コマンドコード ＝ 0

リクエストヘッダ

位　置	サイズ	I/O	要　　求
0 (A5)	1B	−	26 (定数)
1 (A5)	1B	−	未使用
2 (A5)	1B	I	コマンドコード (0)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)
5 (A5) ⋮ 12 (A5)	8B	−	未使用
13 (A5)	1B	O	ユニット数
14 (A5)	1L	O	デバイスドライバの終りアドレス
18 (A5)	1L	I	パラメータのポインタ
		O	BPB テーブルのポインタ
22 (A5)	1B	I	割りあてられたドライブ番号 (0＝A：)

キャラクタデバイスでの処理に加え、そのデバイスがサポートしているユニットの数と、そのユニット数分の BPB(BIOS パラメータブロック)のアドレスを並べた BPB テーブルのポインタをセットして戻ります。
自分のドライブ番号が渡せるため、タイトルの表示などに使用できます。

●ディスク交換チェック

コマンドコード ＝ 1

リクエストヘッダ

位　置	サイズ	I/O	要　　求
0 (A5)	1B	—	26 (定数)
1 (A5)	1B	I	ユニット番号
2 (A5)	1B	I	コマンドコード (1)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)
5 (A5) ⋮ 12 (A5)	8B	—	未使用
13 (A5)	1B	I	メディアバイト
14 (A5)	1B	O	データバッファ

このファンクションは、システムの内部のI/Oバッファのデータが有効であるかを知るために呼び出されます。

ユニット番号で指定されたドライブのディスクが交換されたか否かをチェックして、

　−1 ＝ ディスクが交換された場合

　　1 ＝ ディスクが交換されていない

の値をデータバッファにセットします。

交換されたか否かが不明な場合は、−1を返すようにしてください。

現在は、メディアバイトは使用していません。

●入力または出力

コマンドコード ＝ 4、8、9

リクエストヘッダ

位 置	サイズ	I/O	要 求
0 (A5)	1B	−	26（定数）
1 (A5)	1B	I	ユニット番号
2 (A5)	1B	I	コマンドコード (4、8、9)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)
5 (A5) ⋮ 12 (A5)	8B	−	未使用
13 (A5)	1B	I	メディアバイト
14 (A5)	1L	I	転送バッファのアドレス
18 (A5)	1L	I	セクタ数
22 (A5)	1L	I	開始セクタ番号

ユニット番号で指定されたドライブの開始セクタ番号から指定セクタ分のデータを、転送バッファに読み込みます（入力 ＝ コマンドコード 4）。

または転送バッファからセクタへ書き出します（出力 ＝ コマンドコード 8）。

コマンドコード 9 の場合は、書き出した後にそのデータを読み込んで、実際に書き出せていることを確認します（ベリファイ）。

●ドライブコントロール & センス

コマンドコード ＝ 5

リクエストヘッダ

位　置	サイズ	I/O	要　　求
0 (A5)	1B	－	26（定数）
1 (A5)	1B	I	ユニット番号
2 (A5)	1B	I	コマンドコード (5)
3 (A5)	1B	O	エラーコード (low)
4 (A5)	1B	O	エラーコード (high)
5 (A5) ： 12 (A5)	8B	－	未使用
13 (A5)	1B	I O	内部コマンド ドライブの状態

ユニット番号で指定されたドライブについて、内部コマンドで指定された処理を行い、ドライブの状態をバッファにセットして戻ります。

内部コマンド

0 ： ドライブの状態を調べるのみ
1 ： イジェクトする（イジェクト禁止の状態の場合は無視）
2 ： イジェクト禁止 1 (ユーザーで使用)
3 ： イジェクト許可 1 (禁止 1 を解除。ユーザーで使用)
4 ： ディスクがセットされていないとき、LED 点灯
5 ： ディスクがセットされていないとき、LED 消灯
6 ： イジェクト禁止 2 (OS 内部で使用)
7 ： イジェクト許可 2 (禁止 2 を解除。OS 内部で使用)

ドライブの状態

bit7 ： 1 ＝ LED 点灯
bit6 ： 1 ＝ EJECT 禁止
bit5 ： 1 ＝ BUFFER あり
bit4 ： 1 ＝ USER 禁止
bit3 ： 1 ＝ ライトプロテクト
bit2 ： 1 ＝ ドライブノットレディ
bit1 ： 1 ＝ メディア挿入
bit0 ： 1 ＝ メディア誤挿入

6.7 BPBテーブル

ブロックデバイスでは、ドライバの初期化時に、そのドライバが利用するディスクの特性をシステムに伝えるために BPB (BIOS パラメータブロック) と呼ぶデータを渡します。
システムはこのデータから、DPB（ドライブパラメータブロック）と呼ばれる管理テーブルを内部で作成し、ドライブの管理をします。
BPB はユニットごとに必要であるため、各 BPB のアドレスをユニット数分並べて BPB テーブルを作成し、このアドレスをシステムに渡すデータとします。

BPB の内容（12バイト）

サイズ	内　容
1W	1セクタあたりのバイト数
1B	1クラスタあたりのセクタ数
1B	FAT 領域の個数
1W	予約領域のセクタ数
1W	ルートディレクトリに入るエントリ数
1W	全領域のセクタ数
1B	メディアバイト
1B	1個の FAT 領域に使用するセクタ数

参考：　DPBの内容（94バイト）

サイズ	内　　容
1B	装置番号　0 = A：　1 = B：
1B	ドライバ内でのユニット番号
1W	1セクタあたりのバイト数
1B	1クラスタあたりのセクタ数−1
1B	先頭クラスタのセクタ番号
1W	FATの先頭のセクタ番号
1B	FAT領域の個数
1B	1個のFAT領域に使用するセクタ数
1W	ルートディレクトリに入るエントリ数
1W	データ部の先頭セクタ番号
1W	総クラスタ数+1
1W	ルートディレクトリの先頭セクタ番号
1L	デバイスドライバへのポインタ
1B	メディアバイト
1B	DPB使用フラグ（−1でアクセスなし）
1L	次のDPBへのポインタ
1W	カレントディレクトリのクラスタ番号
64B	カレントディレクトリの文字バッファ

6.8 サンプルプログラム

●キャラクタデバイス

```
*****************************************************************
*       プリンタドライバ
*
*       -作成方法-
*       as prndrv
*       lk /oprndrv.sys prndrv
*****************************************************************
        .include        doscall.mac
        .text
reqlen  = 0
unitcd  = 1
comcod  = 2
errlow  = 3
erhigh  = 4
***8byte
**init
mxunit  = 13
devend  = 14
bpbpoi  = 18
**read/write
diskid  = 13
dmaadr  = 14
dmalen  = 18
**get
getdat  = 13

prntbl: dc.l    lpttbl
        dc.w    $8000
        dc.l    prnstr
        dc.l    prnent
        dc.b    'PRN     '
prnreq: dc.l    0
prnjmp: dc.l    prnini
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    prnout
        dc.l    prnout
        dc.l    prnosn
        dc.l    okcom
        dc.l    notcom
lpttbl: dc.l    -1
        dc.w    $8000
        dc.l    lptstr
        dc.l    lptent
        dc.b    'LPT     '
lptreq: dc.l    0
lptjmp: dc.l    lptini
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
```

```
        dc.l    notcom
        dc.l    notcom
        dc.l    lptout
        dc.l    lptout
        dc.l    prnosn
        dc.l    okcom
        dc.l    notcom

prnstr: move.l  a5,prnreq
        rts

prnent: movem.l d0/a4-a5,-(sp)
        move.l  prnreq,a5
        lea     prnjmp,a4
entry:  clr.l   d0
        move.b  comcod(a5),d0
        add.w   d0,d0
        add.w   d0,d0
        add.l   d0,a4
        move.l  (a4),a4
        jsr     (a4)
        move.b  d0,errlow(a5)
        lsr.w   #8,d0
        move.b  d0,erhigh(a5)
        movem.l (sp)+,d0/a4-a5
        rts
**
notcom: move.w  #$5003,d0
        rts
**
okcom:  clr.w   d0
        rts


lptstr: move.l  a5,lptreq
        rts

lptent: movem.l d0/a4-a5,-(sp)
        move.l  lptreq,a5
        lea     lptjmp,a4
        bra     entry

prnout:
        movem.l d1/d3/a1,-(sp)
        move.l  dmalen(a5),d3
        beq     prnote
        move.l  dmaadr(a5),a1
prnotl: clr.w   d1
        move.b  (a1)+,d1
        moveq.l #$3f,d0
        trap    #15
        subq.l  #1,d3
        bne     prnotl
prnote: movem.l (sp)+,d1/d3/a1
        bra     okcom

lptout:
        movem.l d1/d3/a1,-(sp)
        move.l  dmalen(a5),d3
        beq     prnote
        move.l  dmaadr(a5),a1
lptotl: clr.w   d1
        move.b  (a1)+,d1
        moveq.l #$3e,d0
        trap    #15
        subq.l  #1,d3
        bne     lptotl
        bra     prnote
```

```
prnosn:
        moveq.l #$3d,d0
        trap    #15
        tst.b   d0
        bne     okcom
        moveq.l #1,d0
        rts
******
prnini:
lptini:
        movem.l d1-d3/a0-a3,-(sp)
        move.l  bpbpoi(a5),a4
                * DEVICE=prndrv.sys #/w96 /l64 /m1
                * *******^=a4
sysskp: move.b  (a4)+,d0
        bne     sysskp
        moveq.l #-1,d1
igetac: move.b  (a4)+,d0
        beq     normal
        cmp.b   #'#',d0
        bne     igetac
nxparg: move.b  (a4)+,d0
        beq     normal
        cmp.b   #'/',d0
        bne     inierr
swchk:  move.b  (a4)+,d0
        and.w   #$df,d0
        cmp.b   #'W',d0
        beq     width
        cmp.b   #'M',d0
        beq     modset
        cmp.b   #'L',d0
        bne     inierr
        bsr     numinp
        tst.b   d0
        bne     inierr
        tst.b   d2
        beq     inierr
        cmp.w   #257,d2
        bcc     inierr
        subq.w  #1,d2
        ror.w   #8,d1
        move.b  d2,d1
        rol.w   #8,d1
        bra     nxparg
width:  bsr     numinp
        tst.b   d0
        bne     inierr
        tst.b   d2
        beq     inierr
        cmp.w   #257,d2
        bcc     inierr
        subq.w  #1,d2
        move.b  d2,d1
        bra     nxparg
modset: bsr     numinp
        tst.b   d0
        bne     inierr
        cmp.l   #32,d2
        bcc     inierr
        move.b  d2,modflg
        bra     nxparg
```

```
normal:
        moveq.l #$3c,d0
        trap    #15
        lea     CZ8PKA,a1
        moveq.l #$3a,d0
        trap    #15
        pea     mes0
        dc.w    _PRINT
        addq.l  #4,sp
        move.l  #prnini,devend(a5)
        movem.l (sp)+,d1-d3/a0-a3
        clr.l   d0
        rts

inierr: pea     mes2
        dc.w    _PRINT
        addq.l  #4,sp
        movem.l (sp)+,d1-d3/a0-a3
        move.w  #$700d,d0
        rts

numinp: clr.l   d2
numinl: move.b  (a4)+,d0
        cmp.b   #'0',d0
        bcs     numine
        cmp.b   #'9'+1,d0
        bcc     numine
        and.w   #$f,d0
        mulu.w  #10,d2
        add.w   d0,d2
        bra     numinl
numine: rts

mes0:   dc.b    13,10
        dc.b    'PRINTER DRIVER for X68000 version 0.50',13,10
        dc.b    'PRN/LPTのファイル名でプリンタ'
        dc.b    'に印字可能です',13,10,0
mes2:   dc.b    13,10
        dc.b    'パラメータに誤りがあります。',0

        .even
CZ8PKA: dc.l    $e00000                 * vram0 address/vram1=+$20000
        dc.w    42-1                    * y dot/12-1
        dc.l    $ffff00                 * last 16bit out
        dc.w    21-1                    * y dot/24-1
        dc.l    $ff0000                 * last 8bit out
        dc.w    96-1                    * x dot/8-1
        dc.b    6,$1b,'K',$1c,'S',6,6,0
        dc.b    2,$1b,'H',0,0,0,0,0
        dc.b    2,13,10,0               * lf code
        dc.b    4,$1b,'%9',15,0         * 24 dot lf
        dc.b    4,$1b,'%9',0,0          * nomal lf
        dc.b    4,$1b,'%9',10,0         * last 16 dot lf
        dc.b    4,$1b,'%9',5,0          * last 8 dot lf
        dc.b    4,$1b,'J',6,0,0,0,0     * $600 dot out
        dc.b    4,$1b,'J',3,0,0,0,0     * $300 dot out
        dc.b    4,$1b,'J',0,18,0,0,0    * 18 dot out
        dc.b    4,$1b,'J',0,36,0,0,0    * 36 dot out
        dc.b    0                       * 0=MSB-LSB     1=LSB-MSB
        dc.b    3                       *
        dc.b    6                       *
modflg: dc.b    7                       * 1/3/7/15/31
```

●ブロックデバイス

```
*****************************************************************
*       SRAMDISKドライバ
*
*       DEVICE = sramdisk.sys をCONFIG.SYSに登録して
*       SWITCH SRAM=Rを実行後、再起動すると
*       SRAMの15Kバイトがディスクとして使えます。
*
*       -作成方法-
*       as sramdisk
*       lk /osramdisk.sys sramdisk
*****************************************************************
        .include        doscall.mac
        .text

reqlen  = 0
unitcd  = 1
comcod  = 2
errlow  = 3
erhigh  = 4
***8byte
**init
mxunit  = 13
devend  = 14
bpbpoi  = 18
bdevno  = 22
**media check
diskid  = 13
diskfg  = 14
**read/write
*diskid = 13
dmaadr  = 14
dmalen  = 18
starec  = 22+2
**get
getdat  = 13

SRAM    = $ed0000
SRAMMD  = $ed002d
SYS00d  = $e8e00d

dsktbl: dc.l    -1
        dc.w    $0000
        dc.l    dskstr
        dc.l    dskent
        dc.b    1,'S_RAM  '
dskreq: dc.l    0
dskjmp: dc.l    dskini
        dc.l    mediac
        dc.l    notcom
        dc.l    notcom
        dc.l    dskinp
        dc.l    dskctrl
        dc.l    notcom
        dc.l    notcom
        dc.l    dskout
        dc.l    dskotv
        dc.l    notcom
        dc.l    notcom
        dc.l    notcom
```

```
dskstr: move.l  a5,dskreq
        rts

dskent: movem.l d0/a4-a5,-(sp)
        move.l  dskreq,a5
        lea     dskjmp,a4
entry:  clr.l   d0
        move.b  comcod(a5),d0
        add.w   d0,d0
        add.w   d0,d0
        add.l   d0,a4
        move.l  (a4),a4
        jsr     (a4)
        move.b  d0,errlow(a5)
        lsr.w   #8,d0
        move.b  d0,erhigh(a5)
        movem.l (sp)+,d0/a4-a5
        rts
**
notcom: move.w  #$5003,d0
        rts
**
mediac: clr.w   d0
        cmp.b   #$f9,SRAM+$400
        beq     medchg
        moveq.l #-1,d0
medchg: move.b  d0,diskfg(a5)
okcom:  clr.w   d0
        rts

dskinp: movem.l d1/a0-a2,-(sp)
        lea     dskins,a0
        bsr     dskpar
        movem.l (sp)+,d1/a0-a2
        rts

dskotv: bsr     dskout
        tst.w   d0
        bne     srmver
        movem.l d1/a0-a2,-(sp)
        lea     srmvfs,a0
        bsr     dskpaw
        movem.l (sp)+,d1/a0-a2
srmver: rts

dskout:
        movem.l d1/a0-a2,-(sp)
        move.b  #$31,SYS00d         * SRAM書き込み許可
        lea     dskots,a0
        bsr     dskpaw
        move.b  #0,SYS00d           * SRAM書き込み禁止
        movem.l (sp)+,d1/a0-a2
        rts

dskctrl:
        move.b  #$42,d0
        cmp.b   #$f9,SRAM+$400
        beq     okrdy0
        bset.l  #2,d0
okrdy0: move.b  d0,getdat(a5)
        bra     okcom
```

```
dskpar:
        cmp.b   #$f9,SRAM+$400
        bne     baddrv
dskpaw: clr.l   d0
        move.w  starec(a5),d0
        add.w   dmalen+2(a5),d0
        cmp.w   rdmaxr,d0
        bhi     rdrcov
        move.l  dmalen(a5),d0
        asl.l   #8,d0
        asl.l   #2,d0
        subq.l  #1,d0
        move.l  dmaadr(a5),a1
        clr.l   d1
        move.w  starec(a5),d1
        lea     SRAM,a2
        asl.l   #8,d1
        asl.l   #2,d1
        add.l   d1,a2
        jsr     (a0)
        rts
baddrv: move.w  #$5007,d0
        rts
rdrcov: move.w  #$5008,d0
        rts

dskins:
dskin1: move.b  (a2)+,(a1)+
        dbra    d0,dskin1         * d0<$8000
        clr.l   d0
        rts

dskots:
dskot1: move.b  (a1)+,(a2)+
        dbra    d0,dskot1         * d0<$8000
        clr.l   d0
        rts

srmvfs:
        cmp.b   (a1)+,(a2)+
        bne     srmvfe
        dbra    d0,srmvfs         * d0<$8000
        clr.l   d0
        rts
srmvfe: move.w  #$700b,d0
        rts

bpbtbl:
        dc.w    1024
        dc.b    1
bpbftc: dc.b    1
        dc.w    1
bpbdrc: dc.w    32
rdmaxr: dc.w    $10               * $80000/$400
bpbfid: dc.b    $f9
bpbfsz: dc.b    1

inittbl:
        dc.l    bpbtbl
```

```
**********************
dskini:
        movem.l d1-d7/a0-a6,-(sp)
        cmp.b   #1,SRAMMD       * SWITCH SRAM=R
        bne     parer2
        move.b  #1,mxunit(a5)
        move.l  #inittbl,bpbpoi(a5)
        move.l  #dskini,devend(a5)
        move.b  bdevno(a5),d0
        add.b   #'A',d0
        move.b  d0,devmes
        pea     mes
        dc.w    _PRINT
        addq.l  #4,sp
        cmp.b   #$f9,SRAM+$400
        bne     clrok
        moveq.l #2,d0
        trap    #15
        btst.l  #0,d0           * shift key on/off ?
        beq     noclr
clrok:  bsr     fatdir
        pea     mesini
        bra     prend
noclr:  pea     mesnoi
prend:  dc.w    _PRINT
        addq.l  #4,sp
        movem.l (sp)+,d1-d7/a0-a6
        bra     okcom

parer2: pea     mes2
errprt: dc.w    _PRINT
        addq.l  #4,sp
        movem.l (sp)+,d1-d7/a0-a6
        move.w  #$700d,d0
        rts


fatdir: lea     SRAM+$400,a0
        move.b  #$31,SYS00d     * SRAM書き込み許可
        move.w  #2*1024/4-1,d0
drcl00: clr.l   (a0)+
        dbra    d0,drcl00
        move.l  #$f9ffff00,SRAM+$400
        lea     SRAM+$800,a1

        dc.w    _GETTIME
        ror.w   #8,d0
        move.w  d0,dirtim
        dc.w    _GETDATE
        ror.w   #8,d0
        move.w  d0,dirdte

        lea     dirdat,a0
        moveq.l #32/4-1,d0
drcplp: move.l  (a0)+,(a1)+
        dbra    d0,drcplp
        move.b  #0,SYS00d       * SRAM書き込み禁止
        rts
```

```
dirdat: dc.b     'SRAM_DISK  ',$08,0,0,0,0
        dc.b     0,0,0,0,0,0
dirtim: dc.b     $A0,$60
dirdte: dc.b     $E7,$0A
        dc.b     0,0,0,0,0,0

mes:    dc.b     13,10
        dc.b     'S-RAM DISK DRIVER for X68000 version 1.00',13,10
        dc.b     'RAMディスク('
devmes: dc.b     'C: $ED0400-15K',0
mesini: dc.b     ')を初期化しました',13,10,0
mesnoi: dc.b     ')は初期化しません',13,10,0
mes2:   dc.b     13,10
        dc.b     'S-RAMは RAMDISKとして使用できません。',0
        end
```

第7章

プログラム
インターフェイス

ユーザープログラム起動後の状態
メモリ管理ポインタ
プロセス管理ポインタ
エラー処理ルーチン

7.1 ユーザープログラム起動後の状態

システムからユーザープログラムが起動した場合には、ユーザープログラムに制御が渡される前に、実行に必要な各種パラメータがレジスタにセットされます。
そのため、一般的な使用では、後に説明する「メモリ管理ポインタ」、「環境の構造」、「プロセス管理ポインタ」などの内容は、あまり参照する必要がありません。

●レジスタ

A0	=	プログラムのメモリ管理ポインタのアドレス
A1	=	プログラム(DATA、BSSを含む)の終わり+1のアドレス
A2	=	プログラムに渡されたコマンドラインのアドレス
A3	=	環境のアドレス
A4	=	プログラムの実行先頭アドレス
SR	=	モード（ユーザーモード）
USP	=	親のスタックそのまま
SSP	=	システムのスタック
そのほか	=	不定

●メモリ

この時点では、確保できる最大のメモリが割りあてられています。
そのためsetblock（$FF4A）によってメモリを解放しなければ、malloc（$FF48）はできません。
このプロセスに与えられたメモリのエンドアドレスは、8(A0)にセットされています（メモリ管理ポインタ内)。
チャイルドプロセスを実行しないならばA0、A1レジスタの内容は不要です。

●コマンドライン

プログラム起動時に指定されたコマンドラインのうち、コマンド名より後ろのオプションパラメータだけが切り出され、A2レジスタによって示されるワークエリアに格納されています。
スペース、タブなどの文字はそのままです。

'>'、'<'、'|'以降の文字はリダイレクト（パイプ）でシステムが処理するため、このエリアには含まれません。

●コマンドラインの構造

先頭1バイトにより、文字列の長さ（バイト数）を示しています。
コマンドラインの終わりには0がつけられます。

例

```
+0    dc.b    7
+1    dc.b    '−W test', 0
```

●環境

A3レジスタによって与えられる環境は、親の環境そのものであり、setenv（$FF52）によって環境の内容を変更した場合、親の環境も変更されることになります。
チャイルドプロセスのみを異なる環境の下で実行させたい場合には、ユーザープログラムが別のエリアにメモリを確保し、新たな環境を用意しなければなりません。
なお、getenv($FF53)やsetenv($FF52)だけを使用するならば、A3レジスタの内容は不要です。

●環境の構造

先頭の4バイトにより、環境エリアの大きさ（無効な文字も含んだ全体のバイト数）を示します。
各環境変数は、1バイトの0によって区切られます。
環境文字列の終端には、さらにもう1バイトの0がつけられます。

例

```
+0    dc.l    環境の長さ（この4バイトも含む環境のバイト数）
+4    dc.b    'path＝a：¥' ,0（0で一つの文字列の終わり）
      dc.b    データ文字列＋0
      dc.b    0    すべての文字列の終わり
      dc.b    ??     残りには何が入っているか不定
```

7.1 ユーザープログラム起動後の状態

●サンプルプログラム

```
        .include        doscall.mac
        .text
pstart:
        lea     usersp,sp       * スタックは必ず設定すること

                                * mallocでメモリを確保するさいには、
                                * まず自分の後についているメモリを開放する
        lea     $10(a0),a0      * メモリ管理のデータ部のポインタを求める
        sub.l   a0,a1           * このプログラムの長さを求める
        move.l  a1,-(sp)
        move.l  a0,-(sp)
        dc.w    _SETBLOCK       * $FF4A
        addq.l  #8,sp
                                * 必要ならメモリを確保して、
                                * そのポインタから使う
        move.l  #$ffffff,-(sp)  * 最大必要なバイト数を指定する
        dc.w    _MALLOC         * $FF48
        addq.l  #4,sp
        tst.l   d0
        bpl     mempok          * 必要なバイト数が確保できたらそれでよい
        and.l   #$ffffff,d0     * できない場合は、とれるだけとる
        move.l  d0,-(sp)
        dc.w    _MALLOC         * $FF48
        addq.l  #4,sp
        tst.l   d0
        bmi     memerr          * メモリ確保できなかった
mempok: move.l  d0,memptr       * 管理ポインタをワークにしまう
        *
        *       プログラム本体
        *
* ふつうに終了する場合
        dc.w    _EXIT           * $FF00 終了

* 常駐終了したい場合
* すべてのメモリを確保した状態では、常駐終了してはならない
* 常駐終了では、trap10からtrap14のベクタはもとに戻されない
        move.w  #128,-(sp)
        move.l  #pend-pstart,-(sp)
        dc.w    _KEEPPR         *$FF31

* エラーなのでエラーコードをもって終了する場合
memerr: move.w  #1,-(sp)
        dc.w    _EXIT2          * $FF4C エラーの場合は0以外の終了コード

        .data
memptr: ds.l    1               * メモリ管理のデータ部ポインタ用ワーク

        .bss
        ds.l    256     * (+1024)
                        * スーパーバイザの場合、
                        * システム用に4Kバイトは必要
usersp:
pend:
        end
```

7.2 メモリ管理ポインタ

1つのプロセスは複数のメモリブロック（環境、プログラム本体、データ部分など）から構成されています。

Human68k の動作中はメモリに複数のプロセスが存在することになり、メモリブロックも多数存在しています。

メモリ管理ポインタは、プロセス間でメモリの干渉が起こらないように、システムがメモリ資源を管理するために設けられたものです。

各プロセスは malloc（$FF48）を行って、システムにメモリ割りあての要求を行い、新たなメモリブロックを確保したり、setblock（$FF4A）を行って、メモリブロックのサイズを変更したりすることができます。

これらメモリブロックの先頭には、システムによって自動的にメモリ管理ポインタが作られます。

このテーブルは、ポインタのリンクによるリスト構造になっており、システムが管理しているメモリは、すべてこのポインタのリストによって記述されています。

各プロセスが割りあてられたメモリブロックだけをアクセスしており、それ以外のメモリはアクセスしない限り、ユーザープログラムの側では、メモリ管理ポインタの内容について、あまり意識する必要はありません。

メモリ管理ポインタは、次の4ロングワードからできており、OS 内の fstmem が先頭になります。

また、ポインタは必ず最下位4ビットが0（$?????0)のアドレスから始まります。

```
dc.l   ?   一つ前のメモリ管理ポインタ
dc.l   ?   このメモリを確保したプロセスのメモリ管理ポインタ
dc.l   ?   このメモリブロックの終わり+1のアドレス
dc.l   ?   次のメモリ管理ポインタ（0なら終わり）
```

Human68k の動作中は、メモリ上は次のような状態になっています（これはメモリの状態を示すのであり、プログラムとして記述したものではありません)。

```
fstmem：   dc.l    0          前がないことを表す0
           dc.l    0          親がいないことを表す0
           dc.l    usmem0
           dc.l    usmem0     最初は0が入っている

＊＊＊ユーザーエリア
usmem0：   dc.l    fstmem     前はfstmem
           dc.l    fstmem     親はfstmem
           dc.l    usend0     終わりはバイト単位で指定できるので
           dc.l    usmem1     usend0とusmem1は同じとは限らない
           ＊
           ＊      ユーザーの作成したプログラム本体
           ＊
usend0：
＊＊＊データエリア
usmem1：   dc.l    usmem0     前はusmem0
           dc.l    usmem0     親もusmem0
           dc.l    usend1
           dc.l    usmem2
           ＊
           ＊      malloc（$FF48）によってusmem0のプログラムが確
           ＊      保したデータエリア1
usend1：
＊＊＊
usmem2：   dc.l    usmem1     前はusmem1
           dc.l    usmem0     親はusmem0
           dc.l    usend2
           dc.l    0          メモリ管理の終わり
           ＊
           ＊      malloc（$FF48）によってusmem0のプログラムが確
           ＊      保した
           ＊      データエリア2
usend2：
```

7.3 プロセス管理ポインタ

exec ファンクションコールによってプログラムがメモリにロードされた場合、プログラムの直前にメモリ管理ポインタ(16バイト)と、その後ろにプロセスの管理のためのワークエリア(240バイト)が確保されます。

これはおもに Human68k が使用するものですが、起動されたプログラムから参照することもできます。

これらのワークエリアに格納された値は、このプロセスの開始にあたって与えられた条件、およびその時点でのシステムの状態を保存してあるもので、プロセスの終了とともに、そのプロセスを起動した親プロセスに戻る際に使用されます。

＊＊メモリ管理ポインタ(16バイト)　(A0 レジスタ)

$000	1L	1つ前のメモリ管理ポインタ
$004	1L	このメモリを確保したプロセスのメモリ管理ポインタ
$008	1L	このメモリブロックの終わり+1のアドレス
$00C	1L	次のメモリ管理ポインタ (0なら終わり)

＊＊プロセス管理ポインタ(240バイト)

$010	1L	プロセスに与えられた環境のアドレス (＝A3 レジスタ)
$014	1L	プロセスが終了した際の戻りアドレス
$018	1L	プロセスが[CTRL]+[C]により中断された際の戻りアドレス
$01C	1L	プロセスがエラーにより中止された際の戻りアドレス
$020	1L	プロセスに与えられたコマンドラインのアドレス(＝A2 レジスタ)
$024	3L	プロセスのファイルハンドラの使用状況 ($024, $025, $026 の BIT0～7 の順で、使用中＝1となる：Ver 2.00 より以前) ($024, $028, $02C の BIT0～31 の順で、使用中＝1となる：Ver 2.00 以降)

$030	1L	プロセスのbssの先頭アドレス
$034	1L	プロセスのヒープの先頭アドレス(bssと同じ)
$038	1L	プロセスの初期スタックアドレス(ヒープの終わり+1 (=A1レジスタ))
$03C	1L	親プロセスのUSPレジスタの値
$040	1L	親プロセスのSSPレジスタの値
$044	1W	親プロセスのSRレジスタの値
$046	1W	アボート(中止)のときのSRレジスタの値
$048	1L	アボート(中止)のときのSSPレジスタの値
$04C	1L	TRAP10 (POWER OFF or リセット処理ルーチン)
$050	1L	TRAP11 (BREAKキーによるHDOFF処理、そのほかのルーチン)
$054	1L	TRAP12 (COPYキーによるHCOPY処理、そのほかのルーチン)
$058	1L	TRAP13 (CTRL+Cキーによるブレークチェックフラグのセット)
$05C	1L	TRAP14 (エラー表示と、中止・再実行・無視の選択ルーチン)
$060	1L	プロセスのフラグ (0で親あり、−1でOSから起動)
$064	7L	未使用 ($064~$07F)
$080	'A:'	exec ($FF4B) されたファイルのドライブ名
$082	'¥BIN¥', 0	exec($FF4B)されたファイルのパス名
$0C4	'COMMAND.X' ,0	exec ($FF4B) されたファイルのコマンド名
$0DC	9L	未使用 ($0DC~$0FF)
$100		プログラム先頭

注：L：ロングワード
W：ワード
B：バイト

7.4 エラー処理ルーチン

標準の状態ではビジュアルモードでの操作を考慮して、なんらかの原因によってエラーが発生した場合には、画面の中央にメッセージを表示するようになっています。

メッセージは画面に直接出力されるため、コマンドモード時にCTTYコマンドによって別のターミナルからアクセスしている場合には、そのターミナルにはメッセージが表示されません。

標準エラー出力にメッセージを出力するためには、エラーが発生した際に呼ばれるTRAP14処理ルーチンを新たに作成する必要があります。

● TRAP14処理

システムがエラーを検出した場合、D7.Wにエラー番号をセットして、TRAP14が実行されます。

TRAP14処理ルーチンでは、エラーの種別に応じてメッセージを出力した後、キーボードからの入力によって、D7. Wに2（無視）か1（再実行）をセットした後にRTEによって復帰します。

中止の場合は、TRAP15によるアボート処理を行います。

●エラー種別

D7.W	＝$0002	バスエラーが発生した
	＝$0003	アドレスエラーが発生した
	＝$0004	おかしな命令を実行した
	＝$0005	0で除算した
	＝$0006	CHK命令を実行した
	＝$0007	TRAPV命令を実行した
	＝$0008	特権命令を使用した
	＝$0000～$00FF	未登録の割り込み（中止のみ）
	＝$0100～$01FF	未登録のIOCSコール（中止のみ）
	＝$0200～$0FFF	リザーブ
	＝$1000～$1FFF	エラー（中止のみ）
	＝$2000～$2FFF	エラー（再実行のみ）
	＝$3000～$3FFF	エラー（中止 & 再実行）
	＝$301F	NMI　（中止 & 再実行）
	＝$4000～$4FFF	エラー（無視のみ）

=$5000～$5FFF	エラー（中止 & 無視）
=$6000～$6FFF	エラー（再実行 & 無視）
=$7000～$7FFF	エラー（中止 & 再実行 & 無視）
=$8000～$EFFF	リザーブ
=$F000～$FDFF	F系列のDOSコール以外の割り込み（中止のみ）
=$FE00～$FEFF	演算パッケージが組み込まれていない（中止のみ）
=$FF00～$FFFF	未登録のDOSコール（中止のみ）
??01	無効なユニット
??02	ドライブの準備ができていない
??03	無効なコマンドである
??04	CRCエラー
??05	ディスクの管理領域がこわれている
??06	シークエラー
??07	無効なメディア
??08	セクタが見つからない
??09	プリンタオフライン
??0A	書き込みエラー
??0B	読み込みエラー
??0C	そのほかのエラー
??0D	ライトプロテクト（とり替え不可）
??0E	書き込み不可（とり替え可能。ディスクが入っていないときに戻ることあり）

7.4　エラー処理ルーチン

●エラー処理ルーチン

```
*****************************************************************
*****************************************************************
**********      ctty=auxしても使えるエラー処理          ************
**********      ルーチン。常駐終了します。            ************
*****************************************************************
*****************************************************************
        .include        doscall.mac
        .text

stapr:  bra     endpr                   * エラー処理アドレスの登録

errjob: movem.l d0-d6/a0-a6,-(sp)       * D7/A7以外のレジスタ保存
        bsr     errprt                  * メッセージ出力
        lea     crlfms,a1
        bsr     a1mess
        lea     dose40,a1
        bsr     a1mess
        ror.w   #8,d7                   * 上位8ビットチェック
        cmp.b   #$80,d7
        bcc     aborto                  * $8000～$FFFF……中止のみ
        cmp.b   #$10,d7
        bcc     nabort                  * $0000～$0FFF……中止のみ
aborto: move.b  #$10,d7                 * 中止のみフラグセット
nabort: lea     abtmes,a1
        btst.l  #4,d7                   * 中止メッセージ
        bne     skpms0
        lea     spcmes,a1
skpms0: bsr     a1mess
        lea     dose41,a1
        bsr     a1mess
        lea     rtymes,a1
        btst.l  #5,d7                   * 再実行メッセージ
        bne     skpms1
        lea     spcmes,a1
skpms1: bsr     a1mess
        lea     dose42,a1
        bsr     a1mess
        lea     ignmes,a1
        btst.l  #6,d7                   * 無視メッセージ
        bne     skpms2
        lea     spcmes,a1
skpms2: bsr     a1mess
        lea     dose43,a1
        bsr     a1mess
        and.w   #$f8ff,sr
skpms3: bsr     keyget          * キーの読み込み
        and.w   #$df,d0
        cmp.b   #'A',d0         * A....中止
        beq     abortk
        cmp.b   #'R',d0         * R....再実行
        beq     retryk
        cmp.b   #'I',d0         * Iでなければループ
        bne     skpms3
musik:  btst.l  #6,d7
        beq     skpms3
        move.b  #2,d7           * 無視してよければD7 = 2
        bra     keyoke
```

```
retryk: btst.l  #5,d7
        beq     skpms3
        move.b  #1,d7           * 再実行ならD7 = 1
        bra     keyoke
abortk: btst.l  #4,d7
        beq     skpms3
        clr.b   d7              * 中止する場合はD7 = 0
keyoke: lea     crlfms,a1
        bsr     a1mess
        tst.b   d7
        beq     abortj
keyokr: movem.l (sp)+,d0-d6/a0-a6       * レジスタを復帰する
        rte                     * D7を持って返る

abortj: move.w  #$ff,d0         * アボート処理
        trap    #15
        bra     keyokr

errprt:
        lea     errnmi,a1
        cmp.w   #$1f,d7
        beq     nmiprt          * $??1F..NMI
        cmp.w   #$301f,d7
        beq     nmiprt          * $301F..NMI
        cmp.w   #$2,d7
        bcs     iocser          * $0000～$0001……未定義
        cmp.w   #9,d7
        bcs     meser0          * $0002～$0008……バスエラーそのほか
        cmp.w   #$1000,d7
        bcs     iocser          * $0009～$0FFF……未定義
        cmp.w   #$8000,d7
        bcc     iocser          * $8000～$FFFF……未定義
        clr.l   d0
        move.b  d7,d0
        subq.l  #1,d0
        cmp.b   #$0e,d0
        bcc     iocser          * $??0F～……未定義
        lea     errms7,a1
        bra     meserp
meser0: clr.l   d0              * レジスタ表示ありのメッセージ
        move.b  d7,d0
        subq.w  #2,d0
        lea     errms0,a1
        mulu.w  #29,d0
        add.l   d0,a1
nmiprt: bsr     a1mess
        lea     doser5,a1
        bra     erelse
meserp: mulu.w  #53,d0          * レジスタ表示なしのメッセージ
        add.l   d0,a1
        bra     a1mess
iocser: lea     doser0,a1       * エラー番号の表示
        bsr     a1mess
        move.w  d7,d2
        lea     dtbuf,a1
        bsr     d2hex4
        lea     dtbuf,a1
        bsr     a1mess
        lea     doser1,a1
```

```
erelse: bsr     a1mess              * SR、PCレジスタ表示
        move.w  (a6),d2
        lea     dtbuf,a1
        bsr     d2hex4
        lea     dtbuf,a1
        bsr     a1mess
        lea     doser2,a1
        bsr     a1mess
        move.l  2(a6),d2
        lea     dtbuf,a1
        bsr     d2hex8
        lea     dtbuf,a1
        bsr     a1mess
        lea     doser3,a1
        bra     a1mess
**************************************************
a1mess: move.w  #2,-(sp)            * A1レジスタのメッセージ表示
        move.l  a1,-(sp)
        dc.w    _FPUTS
        addq.l  #6,sp
        rts
****
d2hex8: swap    d2                  * 16進8桁表示
        bsr     d2hex4
        swap    d2
d2hex4: rol.w   #4,d2               * 16進4桁表示
        bsr     d2hex1
        rol.w   #4,d2
        bsr     d2hex1
        rol.w   #4,d2
        bsr     d2hex1
        rol.w   #4,d2
d2hex1: move.w  d2,d1
        and.w   #$f,d1
        cmp.w   #10,d1
        bcs     nohexc
        addq.w  #7,d1
nohexc: add.w   #$30,d1
***
conput: move.b  d1,(a1)+
        clr.b   (a1)
        rts

keyget:
        moveq.l #$fd,d0
        trap    #15                 * アボートフラグのリセット
keygtl: move.w  #2,-(sp)
        dc.w    _FGETC              * 一文字入力
        addq.l  #2,sp
        rts

        .data
```

```
*************************************************************
* メッセージ                                                *
*************************************************************
* エラー($ffff)が発生しました(SR=$2000:PC=$00002000)     *
*         中止<A> 再実行< > 無視< >?                        *
*         中止<A> 再実行<R> 無視<I>?                        *
*************************************************************
doser0: dc.b    ' エラー($',0
doser1: dc.b    ')が発生しました'
doser5: dc.b    '(SR=$',0
doser2: dc.b    ':PC=$',0
doser3: dc.b    ') ',0
dose40: dc.b    '         中止<',0
abtmes: dc.b    'A',0
spcmes: dc.b    ' ',0
dose41: dc.b    '> 再実行<',0
rtymes: dc.b    'R',0
dose42: dc.b    '> 無視<',0
ignmes: dc.b    'I',0
dose43: dc.b    '> ?',0
crlfms: dc.b    13,10,0

*               * エラー($ffff)が発生しました(SR=$2000:PC=$00002000) *
errms0: dc.b    '  バスエラーが発生しました  ',0
        dc.b    'アドレスエラーが発生しました',0
        dc.b    ' おかしな命令を実行しました ',0
        dc.b    '      0で除算しました       ',0
        dc.b    '   CHK命令を実行しました    ',0
        dc.b    '  TRAPV命令を実行しました   ',0
        dc.b    '   特権命令を使用しました   ',0
errnmi: dc.b    'Interpt.Switchが押されました',0
errms7: dc.b    '           無効なユニット番号を指定しました           ',0
        dc.b    '        ディスクが入っていません、入れてください        ',0
        dc.b    '    デバイスドライバに無効なコマンドを指定しました    ',0
        dc.b    '                      CRC エラー                      ',0
        dc.b    ' ディスクの管理領域が破壊されています、使用不能です ',0
        dc.b    '                     シークエラー                     ',0
        dc.b    '              無効なメディアを使用しました              ',0
        dc.b    '                 セクタが見つかりません                 ',0
        dc.b    '              プリンタがつながっていません              ',0
        dc.b    '                    書き込みエラー                    ',0
        dc.b    '                    読み込みエラー                    ',0
        dc.b    '                  エラーが発生しました                  ',0
        dc.b    ' プロテクトをはずして、同じディスクを入れてください ',0
        dc.b    'プロテクトされています、ディスクを取り替えてください',0


dtbuf:  ds.b    10

***エラー処理ルーチンのアドレスを登録する***
        .even
endpr:  pea     errjob
        move.w  #$2e,-(sp)
        dc.w    _INTVCS
        addq.l  #6,sp
        clr.w   -(sp)
        move.l  #endpr-stapr,-(sp)
        dc.w    _KEEPPR

        .end
```

●プロセスを中断した際の戻りアドレス

プロセスをアボートした場合に制御が渡されるアドレスは、プロセスごとに設定できプロセス管理ポインタの一部に格納されています。

この中断アドレスを新たに設定するためには、エラーによる中断の戻りアドレスは、パラメータとして$FFF2、CTRL+Cによる中断の戻りアドレスは、パラメータとして$FFF1を使用してintvcs ($FF25) を行います。

例

```
pea         abort_ job
move. w     #$FFF2, -(sp)
dc. w       _INTVCS
addq. l     # 6,sp
```

第 8 章

Human68k ver. 2. 0 の新機能

バックグラウンド処理とは
コモンエリア
ファイルシェアリングとファイルのロック
オーバーレイ X ファイルとバインド
サンプルプログラムの解説

8.1 バックグラウンド処理とは

Human68k ver. 2. 0では、複数のユーザープログラムを並列実行できる「バックグラウンド処理」をサポートしています。

この機能により、X68000をネットワークのファイルサーバとして動作させる、あるいは、TIMERコマンドのように指定時間にADPCMやTVをコントロールする、といった処理がHuman68kの管理下でできるようになります。

ここでは、バックグラウンド処理の概要について解説します。

バックグラウンド処理プログラムは、ここで解説するDOSコールを組み合せて作成します。

その使用方法については「8. 5　サンプルプログラムの解説」を参照してください。

8.1.1 プロセスとは

Human68kの管理下で実行されるプログラムは、「プロセス」と呼ばれます。

外部記憶装置（フロッピーディスクなど）に格納されているプログラム・ファイルをHuman68kがメモリ上にロードすると、それはプロセスとなります。

各プロセスには他のプロセスとは独立した環境が用意されます。

プロセスはDOSコールのexec（$FF4B）を実行するごとに、1個づつ増加します。

このとき、exec($FF4B)を実行したプロセスを親プロセス、exec($FF4B)によりディスクからメモリへロード・実行されるプロセスを子プロセスと呼びます。

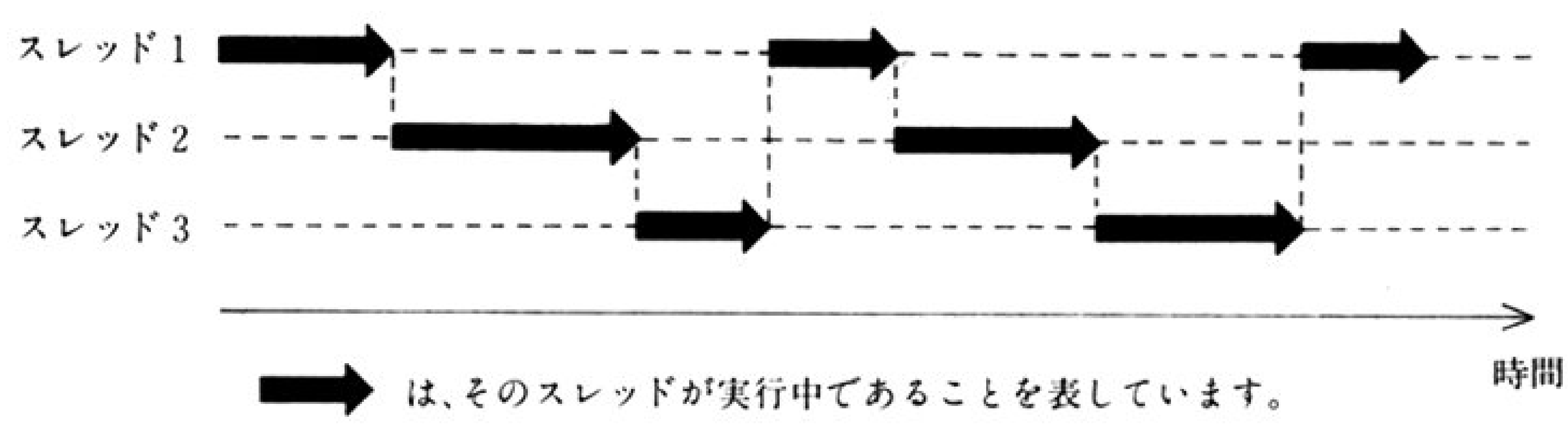
つまり、COMMAND. X上からアプリケーションを実行した場合、COMMAND. Xを親プロセス、そのアプリケーションを子プロセスと呼びます。

メモリ上に複数のプロセスを同時に存在させることは可能ですが、これまで、各プロセスを並列に実行する機能はHuman68kに用意されていませんでした。

しかし、Human68k ver. 2. 0より、次に説明する「スレッド」を利用することにより、各プロセスを並列に実行することができるようになりました。

8.1.2 スレッドとは

スレッドとは、メモリ上でプログラムが実行されるときの制御の流れといえます。
つまり、並列に実行されているプログラムの1つ1つがスレッドなのです。
実際には、CPU は同時に1つの命令しか実行できないので、Human68k が各スレッドの実行を高速に切り替えることで、見かけ上並列に実行されているように制御します（下図参照）。



1つのプロセスを複数のスレッドに分割して並列に実行させることもできますが、通常は1つのプロセスを1つのスレッドとして割り当てるべきです。
なぜなら1つ1つのプロセスから複数のスレッドを実行した場合、どれか1つのスレッドが環境を変更した場合、他のスレッドの環境も変更してしまうからです。
スレッドは、DOS コールの open_pr（$FFF8）を実行することで、システムに登録されます。
open-pr（$FFF8）を実行するごとにスレッドは1個づつ増加します。

8.1.3 スレッドの状態

スレッドの状態には、実行状態であるアクティブ状態と、待機状態であるスリープ状態があります。
あるスレッドが、他のスレッドからの実行要求があるまで何も実行する必要のない場合には、スリープ状態になることにより、要求があるまで待機します。
そうすることで実行時間を余計に消費することが無くなり、全体の実行速度が遅くならずに済みます。
そのための DOS コールが sleep_pr（$FFFC）です。

このDOSコールでは、スリープ状態になっている時間を指定することができます。

8.1.4 スレッド間通信

複数のスレッド間で、情報の受け渡しを行う場合、無秩序に情報の書き込みや読み込みを行うと、複数の書き込みが衝突したり、書き込みの途中で読み込みが行われ、正しく情報が受け渡しできない場合があります。
そのために、情報の受け渡し（通信）を行うためのDOSコールが用意されています。
それが、DOSコールのsend_pr（$FFFD）です。
また、このDOSコールでは、情報の通信以外にも、スリープ状態のスレッドを、アクティブ状態にすることができます。
また、コモンエリアを使って、スレッド間で通信を行うこともできます。
これについては、「8. 2　コモンエリア」で説明しています。

8.1.5 スレッドの区別

複数のスレッドそれぞれには、固有のIDと、プログラム名が割り当てられます。
このIDや名前によって、どのスレッドからどのスレッドへ情報通信が行われるのかを指定します。
したがって、各スレッドのIDや名前を取得することが必要です。
そのためのDOSコールが、get_pr（$FFFA）です。
また、このDOSコールでは、各スレッドの詳細な情報を取得することができます。

8.1.6 並列処理と実行権

並列処理とは、一定の時間間隔でスレッドの実行を切り替えることによって見かけ上、同時にプログラムが実行されているように処理することをいいます。
あるスレッドが一定時間実行している状態のことを、そのスレッドが実行権を持っていると表現します。

このとき、あるスレッドがスーパーバイザ状態になっていると、DOS コールを行うなどしない限りスレッドが切り替わりません（このような状態を「実行権の占有」といいます）。
このような場合、他のスレッドの処理が停滞してしまうので、これを防ぐために一定の周期で実行権を放棄しなければなりません。
そのための DOS コールが change_pr（$FFFF）です。

8.1.7 スレッドの破棄

並列処理中に何らかの理由によって、特定のスレッドが不必要になる場合があります。
また、プロセスの終了時には、そのプロセスが生成したすべてのスレッドを破棄しなければなりません。
そのための DOS コールが kill_pr（$FFF9）です。
スレッドを破棄する場合、スレッドを無秩序に破棄してはいけません。
無秩序に破棄を行うと、システムで回復不可能で致命的な異常が発生する場合があります。
自分以外のスレッドを破棄する場合は、破棄したいスレッドに破棄要求を行います。
その後、すみやかに実行権を放棄して破棄したいスレッドに実行権をわたします。
破棄要求を受け取ったスレッドは、それが受け入れられる要求であれば、要求にしたがって破棄を行います。
受け入れられない要求であれば、その要求を捨てた後、実行権を放棄します。
その後、要求を発行したスレッドに実行権が戻ってきた時点で、要求したスレッドの詳細情報の取得 get_pr（$FFFA）を試みます。
取得が成功すれば、破棄できなかったことを示します。
失敗すれば、破棄できたことを示します。
自分自身のスレッドを破棄したい場合は、まず永久スリープ状態に入り、他のスレッドからアクティブ状態にされるまで待機します。
アクティブ状態にされたら速やかにスレッドを破棄します。
他のスレッドは、どれかのスレッドが上記のような待機状態になっている可能性があるので、定期的に通信を行います。
そしてこのとき、通信に失敗したならば、そのスレッドが自分自身の破棄を行ったことを知ります。

8.1.8 環境の競合

マルチスレッドとは、1つのプロセス上で複数のスレッドが実行されている状態をいいます。
マルチスレッド実行状態では、すべてのスレッドは、同一の環境で実行されます。
同一の環境で実行されるということは、プロセスが同一であり、メモリや、DOSのエラー処理、その他すべて同じ環境で実行されていることを意味しています。
したがって、複数のスレッド内でメモリの確保を行ったり、他のスレッドを残した状態で、DOSコールexit（$FF00）で終了したり、CTRL-CやDOSのエラーアボートで終了したりしてはならないということです。
標準状態では、CTRL-CやDOSのエラーアボート処理で終了するようになっていますので、それらの処理をそれぞれのプロセスで処理することが必要になります。

8.1.9 資源の共用

ファイルや画面やキーボードなど入出力が行われるデバイスでは、各スレッドでの競合の発生を防止する必要があります。
ファイル資源の競合を防ぐには、ファイルシェアリングやファイルロックのためのDOSコールが用意されています。
画面やキーボードについては、スレッドによって競合しないように注意してください。
メモリ資源については、各スレッドで、同一なメモリ管理上で動作します。
スレッド単位でメモリ管理を分割するためのDOSコールを使用しない場合には、無秩序にメモリの確保、開放などを行ってはなりません。

8.1.10 子プロセスの起動

あるスレッドが子プロセスを実行する場合、子プロセスから戻って来た直後で、一度、実行権の放棄を行ってください。
これによって、他のスレッドがそのことを知ります。

8.2 コモンエリア

親プロセスと子プロセス、あるいは2つの子プロセスとの間でデータのやり取りを行うには、いくつかの方法があります。
たとえば、

- 子プロセスに渡されるコマンドラインにパラメータを設定する
- 親プロセスと子プロセスで共通の環境エリアを設ける
- ユーザープログラム中で共有メモリ領域を確保し、通信を行う

しかし、このような方法には以下に示す欠点があります。

- プロセス間で相互に通信するのが難しい
- やり取りされるデータの形式や大きさが限定される
- ユーザープログラムにかかる負担が大きい

Human68k ver. 2.0では、このプロセス間のデータのやり取りをOSレベルでサポートしました。
ユーザープログラムからDOSコールを呼び出すことにより、OSが「コモンエリア」と呼ばれる、各プロセス間で共有されるメモリ領域を確保し、アクセスされるときの排他制御などを行います。
なお、コモンエリアはプロセス間はもちろん、スレッド間でも利用することができます。

8.2.1 コモンエリアの機能

コモンエリアに関して、以下に示す機能があります。

- コモンエリアの確保および削除
- コモンエリアに対するデータの読み込み／書き込み
- コモンエリアのロックおよびロック解除

コモンエリアを確保するときには名前をつけます。
アクセスやロックの際には、コモンエリアにつけられた名前を指定します。

また、名前の他にも、アクセスやロックの対象となる部分も指定します。つまり「?????という名前のコモンエリアの先頭から??バイトより??バイト分だけ読み込む」ということができます。
またコモンエリアは複数個設けることができ、その上限はメモリ容量に依存します。

8.2.2 コモンエリアの利用方法

コモンエリアを利用するには、DOSコールのcommon（$FF55）を用います。
以下の解説は、本マニュアルのDOSコールcommonの項を参照してください。

①コモンエリアの確保

common（2, NAME, POS, BUFFER, LEN）

NAME　：新たに確保するコモンエリアにつける名前
POS　　：初期化するデータを書き込む先頭からの位置
BUFFER：初期化するデータが格納されているアドレス
LEN　　：初期化するデータのバイト数

リターン値は、実際に書き込んだデータのバイト数、またはエラーコード

②コモンエリアの検索

common（0, NAME）

NAME　：検索するモンエリアの名前

リターン値は、コモンエリアが見つかった場合はそのバイト数、見つからなければエラーコード

③コモンエリアからの読み込み

common（1, NAME, POS, BUFFER, LEN）

NAME　：対象のコモンエリアの名前
POS　　：読み込むデータの先頭からの位置
BUFFER：読み込むデータを格納するアドレス
LEN　　：読み込むデータのバイト数

リターン値は、実際に読み込んだデータのバイト数、またはエラーコード

④コモンエリアへの書き込み
common (2, NAME, POS, BUFFER, LEN)
NAME　：対象のコモンエリアの名前
POS　：書き込むデータの先頭からの位置
BUFFER：書き込むデータが格納されているアドレス
LEN　：書き込むデータのバイト数
リターン値は、実際に書き込んだデータのバイト数、またはエラーコード

⑤コモンエリアのロック
common (3, NAME, POS, ID_PSP, LEN)
NAME　：対象のコモンエリアの名前
POS　：ロックする領域の先頭からの位置
ID_PSP：ロックするときにつけるID
LEN　：ロックする領域のバイト数
リターン値は、エラーコード
注：ロックを行うと、ID_PSPをもつプロセスやスレッドの他からはアクセスできなくなります。

⑥コモンエリアのロック解除
common (4, NAME, POS, ID_PSP, LEN)
NAME　：対象のコモンエリアの名前
POS　：ロック解除する領域の先頭からの位置
ID_PSP：ロックしたときにつけたID
LEN　：ロック解除する領域のバイト数
リターン値は、エラーコード

⑦コモンエリアの削除
common (5, NAME)
NAME　：対象のコモンエリアの名前
リターン値は、エラーコード

8.3 ファイルシェアリングとファイルのロック

複数のスレッドが並列に実行される場合、注意すべきこととして、ファイルの取り扱いがあります。
たとえば、あるスレッドがあるファイルをオープンして書き込みを行っている間に、別のスレッドがそのファイルをオープンしてアクセスする、という状況が考えられます。
そのため、Human68k ver. 2. 0では「ファイルシェアリング」という機能を新たに付加しました。
ファイルシェアリングは、ファイルの共有を許す機能です。
また、ファイルをオープンするときに、他からのアクセスを禁止するファイルのロックという機能も加えられました。
これらの機能によりネットワーク環境が実現可能となりました。

8.3.1 ファイルシェアリングの使用方法

シェアリングの指定は、ファイルをオープンするときに指定します。
ファイルのオープンは、DOS コールの open（$FF3D）を使用します。
以下の解説は、本マニュアルの DOS コール open の項を参照してください。

open（NAMEPTR, MD）
NAMEPTR：文字列（ファイル名）へのポインタ
MD：ビット 6～4 ＝ シェアリングモードの設定
　　000 ＝ 互換モード
　　001 ＝ 読み込み／書き込みの両方の禁止
　　010 ＝ 書き込み禁止
　　011 ＝ 読み込み禁止
　　100 ＝ 許可
　ビット 1～0 ＝ 読み書きの指定
　　00 ＝ 読み込みモードでオープン
　　01 ＝ 書き込みモードでオープン
　　10 ＝ 読み込み／書き込みのモードでオープン

リターン値は、オープンしたファイルのファイルハンドルかエラーコード

8.3.2 ファイルのロックの使用方法

ファイルのロックは、DOS コールの lock ($FF5C) を使用します。

lock (MD, FILENO, OFFSET, LEN)

MD	：1 ＝ ファイルのロック
	0 ＝ ファイルのロック解除
FILENO	：ロックするファイルのファイルハンドル
OFFSET	：ロックする部分の先頭からのオフセット
LEN	：ロックするバイト数

8.4 オーバーレイXファイルとバインド

Human68k ver. 2. 0 は、「オーバーレイ X ファイル」という実行可能ファイルをサポートしています。
オーバーレイ X ファイルとは、複数の X 形式実行可能ファイルを 1 つのファイルにまとめたものです。
Human68k ver. 2. 0 は、このオーバーレイ X ファイル中にある任意の X 形式の実行ファイルをメモリにロード・実行することができます。
オーバーレイ X ファイルは、使用可能なメモリ容量に対してプログラムやデータのサイズが大きい場合に使用します。
メモリ不足で実行できないほど大きいプログラムを何とかして実行する 1 つの手段として、プログラムを分割するということが考えられます。
例えば、プログラムを 1 つのメインプログラムと複数のサブプログラムに分割してディスクに格納しておき、実行する際には、メモリ上には必要なプログラムだけをその都度ロードして実行する、という方法があります。
しかしこの場合メインプログラムが、サブプログラムのロード・実行・終了の制御や、サブプログラム同士のデータの受け渡しなどの処理を行わなければならず、プログラミングが難しくなります。
オーバーレイ X ファイルと次に解説する BIND というコマンドは、このように分割されたプログラムの実行を支援するものです。

8.4.1 オーバーレイ X ファイルの作成

オーバーレイ X ファイルを作るには、BIND というコマンドを使います。
BIND の書式は以下の通りです。

BIND　［／O］　＜オーバーレイ X ファイル名＞　＜ファイル名＞　［＜ファイル名＞・・・］　［／D｜／X｜／L］　［／T＜パス名＞］

‘［’と‘］’で囲まれたものは省略が可能です。
また‘｛’で区切られているものは、そのうちの 1 つを選択できます。
／O スイッチは、新たにオーバーレイ X ファイルを作成するときに用います。
たとえば

BIND　/O　sample. x　sample1. x　sample2. x　sample3. x

とすると、sample1. x，sample2. x，sample3. x を1つにまとめたオーバーレイ X ファイル sample. x が作成されます。
すでに存在しているオーバーレイ X ファイルに、X 形式の実行可能ファイルを追加する場合には、以下のように実行します。

BIND　sample. x　sample4. x　sample5. x

その他のスイッチには、以下の機能があります。

- ●/D スイッチ：オーバーレイ X ファイルの中から、指定されたファイルを削除する
- ●/L スイッチ：オーバーレイ X ファイルに含まれているファイル名を表示する
- ●/X スイッチ：オーバーレイ X ファイルの中から、指定されたファイルを取り出す
- ●/T スイッチ：指定されたパスに、テンポラリファイルを作成する

8.4.2 オーバーレイXファイルに含まれているプログラムの実行

オーバーレイ X ファイルに含まれている各ファイルには、モジュール番号が付けられます。
オーバーレイ X ファイル中のファイルを選択するには、このモジュール番号を指定します。
モジュール番号は、オーバーレイ X ファイルに追加された順に0番から付けられます。
前述した BIND コマンドの例では、

sample1. x のモジュール番号は 0
sample2. x のモジュール番号は 1
sample3. x のモジュール番号は 2
sample4. x のモジュール番号は 3
sample5. x のモジュール番号は 4

となります。

コマンドプロセッサ（command. x）からオーバーレイ X ファイル"sample. x"を実行した場合、最初にロード・実行されるのはモジュール番号0の"sample1. x"です。
なお、通常のX形式の実行可能ファイルはモジュール番号が0となっているので、そのままロード・実行されます。
さて、"sample2. x"や"sample3. x"などモジュール番号が0以外のファイルは、直接コマンドプロセッサから実行できません。
つまり、モジュール番号0のファイルがメインプログラムとして動作し、モジュール番号が0以外のファイルがサブプログラムとして必要なときだけ、メインプログラムから呼び出されるわけです。
したがって、モジュール番号が0である"sample1. x"がメインプログラムとして機能し、"sample2. x"や"sample3. x"などのサブプログラムをロード・実行しなければなりません。
そのための DOS コールが exec（$FF4B）です（本マニュアルの DOS コールの解説の exec の項を参照してください。
この exec のパラメータである MD にモジュール番号を指定します。
MD の意味は以下の通りです。

15〜8 ビット：モジュール番号（0〜255）
 7〜0 ビット：コントロールモード
　　0：ロードおよび実行
　　1：ロードのみ
　　2：コマンドパスの検索
　　3：特殊なロード
　　4：実行のみ
　　5：モジュール番号を得る

したがって、"sample1. x"から"sample2. x"をロード・実行するには、MD =$0100 として exec（$FF4B）を呼び出します。
また、ファイル名からモジュール番号を得ることができるので、"sample1. x"は"sample2. x"や"sample3. x"などのサブプログラムのモジュール番号を知らなくてもそれらのプログラムを実行することができます。

8.5 サンプルプログラムの解説

ここでは、サンプルプログラム hide. x を使って、バックグラウンド処理を使用したプログラムの作成方法を解説します。

このプログラムは Human68k ver. 2. 0で新たに付け加えられた、次の機能を使用しています。

・ファイルシェアリング	ファイルの排他制御
・コモンエリア	プロセス、スレッド間の共有記憶領域
・バックグラウンド処理	マルチタスクに近い並列処理
・バインド	オーバーレイ X ファイルの作成

8.5.1 機能と動作環境

●"hide. x"の機能

書式）設定：hide　[ファイル名]
　　　解除：hide　[パスワード]

ファイル名を入力すると、パスワードと時間を聞いてきますので指示にしたがって入力してください。
解除するには、設定時に指定したパスワードを入力してください。

機能）

指定した時間までファイルをロックします。
ロックとはファイルへのアクセスが禁止されることで、ファイルがオープンできなくなります。
リセットするとロックは解除されます。

●"hide. x"の作成方法
リスト 1が、"hide. x"を作成するためのメイクファイル"makefile"です。
実行するときには以下のことを確認してください。

・カレントディレクトリに hidem. s, hidesp. s が存在する

・アセンブラ（AS. X）やリンカ（LK. X）のバージョンが2.00以降である

ソースファイル"hidem. s"，"hidesp. s"はアセンブル・リンクされ、実行可能ファイル"hidem. x"，"hidesp. x"が作成されます。
"hide. x"は"hidem. x"，"hidesp. x"をバインドしたファイルです。
実行するときにはオーバーレイ"hide. x"を実行します。

●システム環境の設定
このプログラムを実行するには、"CONFIG. SYS"において次のような設定をしておく必要があります。

(1)ファイルの共有と排他制御の指定

SHARE ＝ <ファイル数>　<ロック領域数>　［例］SHARE ＝ 10 4

排他制御（ロック）をおこなうファイル数と、ロックする領域数を指定します。
これを設定しないと、ファイルのロックができません。

(2) DOSコールのcommon（$FF55）で使用するメモリ容量の指定

COMMON＝<容量>　［例］COMMON ＝ 32K

DOSコールのcommon（$FF55）で使用するメモリ容量を指定します。
common（$FF55）はプロセスやスレッド間で共有するコモン領域を使用するためのDOSコールです。
これを設定しないと、common（$FF55）は使用できず、呼び出すとエラーになります。

(3)並列処理のための制御情報の設定

PROCESS ＝ <プログラム数>　<レベル>　<タイムスライス値>
［例］PROCESS ＝ 10 2 10

<プログラム数>には、並列に実行するプログラム数を2〜32の範囲で指定します。
<レベル>には、プログラムの実行間隔を決める値を2〜255の範囲で指定

します。
値が小さいほど実行間隔が短くなるので、実行速度が速くなります。
また、<タイムスライス値>には、各プログラムの実行時間を1～100msの範囲で指定します。
PROCESSの指定がないと、並列処理はできません。

●ファイルシェアリング（リスト3　134～143行）
シェアリングは、ファイルの共有化に伴って、最初にファイルをオープンしたユーザーが他のユーザーからのファイルのアクセスを制御するための機能です。
具体的には、DOSコールのopen（$FF3D）のオープンモードが拡張され、重複するファイルの読み書きの禁止、許可を制御することが可能になっています。
このプログラムでは、シェアリングモード$1?を使って、重複するアクセスを排除する例を示しています。

●コモンエリア（リスト3　91～99行，197～202行）
コモンエリアはHuman68kのシステムエリア内に確保され、プロセスあるいはスレッドが共有する記憶領域となります。
たとえて言うなら、プロセス、スレッドの伝言板といえます。
プロセス、スレッドからコモン領域へのアクセスはDOSコールのcommom（$FF55）により行うことができます。
このプログラムでは、コモンエリアにファイルハンドル、パスワードを書き込んでいます。

●バックグラウンド処理
バックグラウンド処理は、見かけ上複数のプログラムを並列に実行させるマルチタスクに近い機能を持っています。
しかし、スレッドが同時にシステムを利用できないなどの制約があります。
実際にバックグラウンドプログラムのを作成する上での注意点として次の2点が上げられます。

(1)スレッド数と同じ数のプログラム（プロセス）を作る必要がある。
1個のプログラムでもバックグラウンド処理を行うことは可能ですが、スレッドが切り替わると環境（プロセス管理領域に設定されているCTRL－CやOSのエラーアボートの処理先など）も切り替わるようにすべきです。
そうしないと、各スレッドが環境を共有することになり、あるスレッドが環

境の変更などを行うと他のスレッドにも影響してしまうからです。
しかし、同一プロセス内では、それらのプロセス管理情報はHuman68kでは1プロセスに1つしか持つことができません。
したがって、実際にそれらを各スレッド毎に切り替えるためには、各スレッドを別の子プロセスとして実行し、スレッドの最初で環境を変更する必要があります。

(2)バックグラウンド処理を行うプログラムは常駐プログラムにする必要があります。
通常のプログラムは、プログラムが終了するとメモリから削除されます。
ところが、常駐プログラムは、プログラムが終了してもその一部をメモリ上に残し、後から何らかの方法で呼び出され、処理を行います。
バックグラウンドプログラムは、親プロセスと子プロセスの2個のプログラムからなり、子プロセスは常駐プログラムになります。
サンプルプログラムでは親プロセスが"hidem. s"、子プロセスが"hidesp. s"に相当します。

8.5.2 親プロセス（リスト2：hidem. s）

親プロセスでは2つの働きを行います。

・スレッドが無ければ、子プロセスを実行してスレッドを生成する。
・スレッドが有れば、スレッド間通信で破棄要求を行う。

親プロセスは次のような構成となります。

(1) DOSのバージョンを調べる（85～92行）
実行可能なDOSのバージョンは2. 00～2. 49です。
DOSコールのvernum（$FF30）を用います。

(2)スレッドの有無を調べる（119～131行）
スレッドがすでに存在していたら(7)へ分岐、存在していなければ(3)へ進みます。
ここではDOSコールのget_pr（$FFFA）を用います。

(3)子プロセスを実行する（17, 21～31行）

ここでは、DOS コールの exec（$FF4B）を用います。
サンプルではオーバーレイファイル中のプロセスを実行しています。
子プロセスは、バックグラウンド処理部をスリープ状態でメモリに常駐させます。

(4)スレッドは永久スリープ状態なのでスレッドをアクティブにするコマンド（$FFFB）を通信する（62～67, 104～118 行）
ここでは、send_pr（$FFFD）を用いて通信を行います。

(5)終了する（20 行）

(6)通信パラメータとして自分とスレッドの ID 番号を求める（119～131 行）
ここでは DOS コールの get_pr（$FFFA）を用います。

(7)スレッドにスレッドの破棄要求（$FFF9）を通信する（65～77 行）
ここでは、send_pr（$FFFD）を用いて通信を行います。
ただし、サンプルプログラムでは、スレッドに破棄要求は行わず、自己破棄を行わせます。

(8)終了する（19～20 行）

8.5.3 子プロセス（リスト3：hidesp. s）

子プロセスは大きく 2 つに分かれています。
バックグラウンド処理を行う常駐部と、スレッドの生成及び常駐終了を行う非常駐部分です。
バックグラウンド処理部分は前に、スレッドの生成、常駐終了部分は後ろに配置されます。
そして、後ろの部分は親プロセスから子プロセスとして実行され、常駐終了時に切り捨てられます。

常駐部は次の働きを行います。

・環境の設定
・スレッド間通信の受信
・スレッドの破棄

・バックグラウンド処理

非常駐部は次の働きを行います。

・スレッドの新規生成
・バックグラウンド処理部のメモリ常駐

常駐部分は次のような構成になります。

⑴環境の設定（14～15, 39～49 行）
スレッドの切り替わりとともに、CTRL－C やエラーアボートの処理先を変更します。
ここでは DOS コールの intvcs（$FF25）を用います。

⑵スレッド間通信の受信（16～38 行）
他のスレッドから通信が来ると通信バッファに必要な情報が格納されます。
またスリープ状態だった場合はアクティブ状態にされます。
スレッド間通信バッファからコマンド、データを読み取り、それに対応した処理プログラムに分岐します。
通信バッファの内容については DOS コールの open_pr（$FFF8）、通信バッファからの読みだしの手順については sleep_pr（$FFFC）の解説を参照してください。
この部分の最後で、DOS コールの sleep_pr（$FFFC）によってスレッドをスリープ状態にしています。
これは、スレッドの処理を軽減し、割り当てられた時間を全て使わずに、システム全体の処理速度を遅くさせないためです。
次の処理が要求されるまで、処理するものがなければ、必ずスリープ状態になるようにしてください。

⑶スレッドの破棄（30 行）
ここでは、DOS コールの kill_pr（$FFF9）を用います
スレッド間通信で破棄要求コマンド（$FFF9）を受信したとき、スレッドはそれが受け入れられる要求であれば、要求にしたがって破棄します。
受け入れられない要求であれば、その要求を捨てた後、実行権を放棄します。
その後、要求を発行したスレッドに実行権が戻ってきた時点で、要求したスレッドの詳細情報の取得を DOS コールの get_pr（$FFFA）で試みます。

取得が成功すれば、破棄できなかったことを示します。
失敗すれば、破棄できたことを示します。
自分自身のスレッドを破棄したい場合は、まず、永久スリープ状態に入り、他のスレッドからアクティブ状態にされるまで待機します。
アクティブ状態にされたら速やかにスレッドを破棄します。
他のスレッドは、どれかのスレッドが上記のような待機状態になっている可能性があるので、定期的に通信を行います。
そしてこのとき、通信に失敗したならば、そのスレッドが自分自身の破棄を行ったことを知ります。
スレッドを破棄する場合、スレッドを無秩序に破棄してはいけません。
無秩序に破棄を行うと、システムで回復不可能な致命的な異常が発生する場合があります。

(4)バックグラウンド処理本体（53～101 行）
ここでは、ユーザーが任意のプログラムを作成できます。
ただし、制約がありますので、後述の注意事項を参照してください。

以上がバックグラウンド処理本体となってメモリに常駐します。
次に、非常駐部を説明します。

(5)スレッドの新規生成（203～216 行）
ここでは、DOS コールの open_pr（$FFF8）を用います。

(6)常駐終了（217～221 行）
ここでは、DOS コールの keeppr（$FF31）を用います。
バックグラウンド処理のデータ、ワーク、スタックエリアを置く位置によって、常駐するプログラム範囲が違います。
つまり、子プロセスの最後にそれらを置くと、プログラム全部を常駐しなければなりません。
しかし、本来この部分は非常駐部分なので、このプログラムのようにデータ、ワークスタックエリアをこのすぐ上、つまり、バックグラウンド処理の最後におきます。
そうすると、常駐範囲はそこまでで良いので、常駐サイズは小さくなります。
このとき、注意することはデータ、スタックなどもテキストセクション（.text）に置かなければならないことです。
したがって、.data, .bss, .stack などのセクション疑似命令をそこには記述

しないように注意してください（下図参照）。

親プロセス	子プロセス
1. スレッドがないときは、子プロセスの非常駐部を実行し、永久スリープ状態の常駐部をアクティブにするよう通信する	—常駐部— メモリに常駐しバックグラウンド処理を行う データをここに書くときは .data, .bss, .stack を入れない
2. スレッドがあるときは、常駐部に破棄要求を通信する	—非常駐部— 常駐部をスレッドとして生成し、常駐終了させる この部分はメモリには残らない

8.5.4 プログラム作成上の注意事項

・子プロセスでプログラムの途中を実行アドレスにする場合、.end 疑似命令の後ろにラベルを書くと、そのラベルから実行が開始されます。

・バックグラウンド処理は、デバッガ（DB. X）でトレースすることができません。
不完全なプログラムをスレッドとして動かすと、止めることが困難でシステムを破壊するおそれさえあります。
したがって開発にあたっては、常駐部と非常駐部にわけてそれぞれが完全に動作するまで十分にデバッグしてください。

・前述のように、バックグラウンド処理には複数のプログラムが必要です。
そこで、BIND コマンドを用いて 1 個のプログラムにまとめます。
BIND コマンドを使う利点として次のことがあります。

(1) 1 個のファイルとして扱えるので取り扱いが便利

(2)子プロセスのファイル名を変更されるおそれがない
(3)子プロセスを実行するとき、自分のバインドファイル名を使っているので、ファイル名を変更しても関係なく実行できる
(4) DOSのバージョンなどの初期環境を調べるルーチンが最初のプログラムだけですむ

バックグラウンド処理プログラムを作成する場合は必ず、BINDコマンドで1個のファイルにしてください。

8.5.5 サンプルプログラム・リスト

リスト1：makefile

```
hide.x: hidem.x hidesp.x
        bind/o hide.x hidem.x hidesp.x
hidem.x: hidem.s
        as hidem.s
        lk hidem.o
hidesp.x: hidesp.s
        as hidesp.s
        lk hidesp.o
```

リスト 2：hidem. s

```
*-----------------------------------------------------------------
*   マニュアルサンプル　親プロセス（ｈｉｄｅｍ．ｓ）
*
*   最初に一度だけ実行され、バックグラウンドプロセスの親プロセスとなる
*   また、スレッドを止めるための、パスワードを送る時に実行する
*
*-----------------------------------------------------------------
        .include        doscall.mac
        .text
        .even
start:
        lea     stkptr,sp           *ユーザースタックの設定
        bsr     version_chk         *ＤＯＳのバージョンチェック
        bsr     cl_cpy              *コマンドラインのコピー
        bsr     get_id              *スレッドはあるか？
        tst     d0
        bmi     exec_pr             *無ければ子プロセスを実行
        bra     pw_send             *有ればパスワードをスレッドに通信
exit_ok:
        DOS     _EXIT
exec_pr:                            *子プロセスの実行
        bsr     m_free
        bsr     psp_read            *自分のファイル名のコピー
        clr.l   -(sp)               *環境
        pea     cl_buff             *コマンド行
        pea     (a0)                *子プロセスファイル名
        move.w  d1,-(sp)            *モード
        DOS     _EXEC
        lea     14(sp),sp
        bsr     exec_chk
        bra     act_pr
m_free:                             *親プロセスの後ろのメモリを解放
        lea     start-$100,a0
        lea     start-$100+$38,a1
        lea     $10(a0),a0
        move.l  (a1),a1
        sub.l   a0,a1
        move.l  a1,-(sp)
        move.l  a0,-(sp)
        DOS     _SETBLOCK
        addq.l  #8,sp
        rts
psp_read:                           *ＰＳＰからファイル名をコピー
        move.b  start-$100+$60,d1
        add.b   #1,d1
        asl.w   #8,d1
        clr.b   d1
        lea     temp_work,a1
        lea     start-$100+$80,a2
        bsr     str_cpy
        lea     start-$100+$c4,a2
        bsr     str_cpy
        lea     temp_work,a0
        rts
exec_chk:
        cmp.l   #$10000,d0
        blt     exec_err
        bsr     get_id              *スレッドのＩＤを求める
        tst     d0
        bmi     nf_err
        rts
act_pr:
        move.l  #0,d5
        move.w  #$fffb,d6           *スレッドをアクティブにする
```

```
        bsr     send_cmd
        tst     d0
        bmi     send_err
        bra     exit_ok
pw_send:
        cmp.b   #4,(a3)             *コマンドラインが4文字か？
        bne     thd_err
        addq.l  #1,a3
        bsr     num_chk             *コマンドラインが数字か？
        move.l  #4,d5               *スレッドにパスワードを送る
        move.w  #$1000,d6
        bsr     send_cmd
        DOS     _CHANGE_PR
        move.l  #0,d5               *スレッドにパスワードを送る
        move.w  #$1000,d6
        bsr     send_cmd
        tst.l   d0
        bpl     send_err
        bra     end_msg
*共通サブルーチン
version_chk:
        DOS     _VERNUM             *DOSバージョンの取得
        cmp.w   #$200,d0            *バージョン1．99以前はエラー
        blt     ver_err
        cmp.w   #$231,d0            *バージョン2．50以上はエラー
        bhi     ver_err             *2．50は3．00相当と判断
        clr.l   d0                  *それ以外を2．00と認識する
        rts
cl_cpy:                             *コマンドラインのコピー
        move.l  start-$100+$20,a2
        move.l  a2,a3
        cmp.b   #0,(a2)
        beq     use_disp
        lea     cl_buff,a1
str_cpy:                            *文字列のコピー
        move.b  (a2)+,(a1)+
        tst.b   (a2)
        bne     str_cpy
        rts
send_wait:
        DOS     _CHANGE_PR
send_cmd:
        move.l  d5,-(sp)            *スレッドへコマンドを送る
        lea     cl_buff,a0
        addq.l  #1,a0
        move.l  a0,-(sp)
        move.w  d6,-(sp)
        move.w  thd_id,-(sp)
        move.w  my_id,-(sp)
        DOS     _SEND_PR
        lea     14(sp),sp
        cmp.l   #-28,d0
        beq     send_wait
        rts
get_id:                             *IDを求めるサブルーチン
        pea     my_tbl              *自分のIDを求める
        move.w  #-2,-(sp)
        DOS     _GET_PR
        addq.l  #6,sp
        move.w  d0,my_id
gets_id:
        pea     thd_tbl             *スレッドのIDを求める
        move.w  #-1,-(sp)
        DOS     _GET_PR
        addq.l  #6,sp
        move.w  d0,thd_id
        rts
num_chk:                            *数字か？
```

```
        move.w  #3,d4
n_cmp:
        cmp.b   #$30,(a3)
        blt     thd_err
        cmp.b   #$39,(a3)+
        bhi     thd_err
        dbra    d4,n_cmp
        rts

*エラーメッセージ
end_msg:        pea     end_mes
                bra     err_msg
exec_err:       pea     exr_msg
                bra     err_msg
nf_err:         pea     nfr_msg
                bra     err_msg
use_disp:       pea     use_msg
                bra     err_msg
par_err:        pea     par_msg
                bra     err_msg
thd_err:        pea     thd_msg
                bra     err_msg
send_err:       pea     sdr_msg
                bra     err_msg
ver_err:        pea     ver_msg
                bra     err_msg
err_msg:        DOS     _PRINT
                addq.l  #4,sp
                bra     exit_ok
                .data
end_mes:        dc.b    'ファイルを開放しました',$0d,$0a,0
nfr_msg:        dc.b    'ファイルを登録できませんでした',$0d,$0a,0
exr_msg:        dc.b    '子プロセスの実行に失敗しました',$0d,$0a,0
use_msg:        dc.b    '使用法：hide［ファイル名］',$0d,$0a
                dc.b    '任意の時間までファイルへのアクセスを禁止します。',$0d,$0a
                dc.b    'パスワード及び時間を聞いてきますので、半角数字で'
                dc.b    '入力してください。',$0d,$0a
                dc.b    '解除するには次のように、コマンドの後にパスワード'
                dc.b    'を入力してください。',$0d,$0a,$0d,$0a
                dc.b    '>hide XXXX  ( X は半角数字 )',$0d,$0a,$0d,$0a,0
thd_msg:        dc.b    '既にファイルが登録されています',$0d,$0a,0
sdr_msg:        dc.b    '通信できません',$0d,$0a,0
par_msg:        dc.b    'パラメータが違います',$0d,$0a,0
ver_msg:        dc.b    'ＤＯＳのバージョンが違います。,$0d,$0a,0
crlf:           dc.b    $0d,$0a,0
temp_work:      ds.b    50
cl_buff:        ds.b    16              *コマンドラインのバッファ
my_id:          dc.w    -1              *自分のＩＤ
thd_id:         dc.w    -1              *スレッドのＩＤ
my_tbl:         ds.b    116             *自分の管理テーブル
thd_tbl:        ds.b    96              *スレッドの管理テーブル
                dc.b    'NEWZ',0        *スレッドの名前
                dc.l    0
                ds.l    1024
stkptr:
                .end    start
```

リスト３：hidesp. s

```
*----------------------------------------------------------------------
*
*  バックグラウンド処理サンプル   子プロセス（ｈｉｄｅｓｐ．ｓ）
*
*    startp から endp までが常駐部（バックグラウンド処理本体）となり、
*   それ以下は一度しか実行されない、非常駐部である
*
*----------------------------------------------------------------------
        .include        doscall.mac
        .text
        .even
*  常駐部の初め
startp:
        move.l  sp,sp_work
        bsr     vect_set        *エラーアボートの処理先の変更
send_chk:
        bsr     time_chk
        cmp.l   #-1,d0
        beq     fin
        lea     mes_buff,a3
        move.w  10(a3),d1       *通信の有無を調べる
        cmp.w   #$-1,d1
        beq     no_send
        move.w  8(a3),d2        *通信コマンドのチェック
        cmp.w   #$1000,d2       *パスワードの受信コマンド
        beq     pw_clr
        cmp.w   #$fff9,d2       *スレッドの破棄要求
        bne     no_com_send
exit:                           *スレッドの破棄
        DOS     _KILL_PR
no_com_send:                    *通信バッファの初期化
        move.w  #0,8(a3)
        move.w  #-1,10(a3)
no_send:
        move.l  #100,-(sp)
        DOS     _SLEEP_PR       *スリープして待機
        addq.l  #4,sp
        bra     send_chk
*  環境変更の例
*
vect_set:                       *アボート時の処理先の設定
        pea     ctrl_c
        move.w  #_CTRLVC,-(sp)
        DOS     _INTVCS
        addq.l  #6,sp
        rts
ctrl_c:                         *エラーアボートルーチン
        move.l  sp_work,sp      *アボート用カレントＳＰ復帰
        bra     no_send         *一旦実行権を放棄してメインへ
*  メッセージ（パスワード）の処理
time_chk:                       *時間との比較
        DOS     _GETTIM2
        tst     d0
        bmi     exit
        move.l  d0,d1
        clr.w   d0
        swap    d0
        cmp.b   hour,d0
        bne     exit_ts
        move.l  d1,d0
        lsr     #8,d0
        cmp.b   min,d0
        bne     exit_ts
        move.l  #-1,d0
```

```
exit_ts:
        rts
pw_clr:
        move.w  #1,d0           *コモンエリアを読み出す
        bsr     com_rw
        lea.l   com_buff,a0
        lea.l   mes_data,a1
        addq.l  #2,a0
        move.l  #3,d1
pw_cmp:                         *パスワードの比較
        cmp.b   (a0)+,(a1)+
        bne     break
        dbra    d1,pw_cmp
        sub.l   #6,a0           *ファイルのクローズ
        move.w  (a0),-(sp)
        DOS     _CLOSE
        addq.l  #2,sp
fin:
        pea     com_name        *コモンネームの破棄
        move.w  #5,-(sp)
        DOS     _COMMON
        addq.l  #6,sp
        DOS     _CHANGE_PR
        bra     exit
break:
        bra     no_com_send
com_rw:                         *コモンの読み書きサブルーチン
        move.l  #20,-(sp)
        pea     com_buff
        move.l  #0,-(sp)
        pea     com_name
        move.w  d0,-(sp)
        DOS     _COMMON
        lea     18(sp),sp
        rts
*スレッドのデータ
*       .data
com_name:
        dc.b    'GT-R',0
*スレッドのワークエリア
*       .bss
hour    ds.b    1
min     ds.b    1
sp_work:        ds.l    1
com_buff:       ds.b    20
mes_buff:                       *スレッド間通信管理バッファ
        dc.l    4               *
        dc.l    mes_data
        dc.w    0
        dc.w    -1
mes_data:                       *実際のメッセージ
        ds.b    4
*       .stack
        ds.l    2048            *スレッド用システムスタック
ssp1:
        ds.l    512             *スレッド用ユーザースタック
usp1:
endp:
*  ここから非常駐
*       .text
main:
        lea     userp,sp
        lea     com_buff,a0     *コモンバッファのポインタ
        bsr     fopen           *ファイルのオープン
        bsr     get_pw          *パスワードの入力
        bsr     get_tm          *時間の入力
        bsr     com_w           *コモンエリアに書き込む
        bsr     thred_open      *スレッドの新規生成
```

```
        bra     keep            *常駐終了
fopen:
        move.w  #$10,-(sp)      *シェアリングによりファイルアクセスを無効にする
        addq.l  #1,a2
        move.l  a2,-(sp)
        DOS     _OPEN
        addq.l  #6,sp
        tst     d0
        bmi     open_err
        move.w  d0,(a0)+
        rts
get_pw:
        lea     pw_msg,a1       *パスワード入力
        bsr     disp
        move.w  #$39,d3
        move.w  #3,d1
i_pw:   bsr     i_num
        add.b   #$30,d5
        move.b  d5,(a0)+
        dbra    d1,i_pw
        lea     crlf,a1
        bsr     disp
        rts
get_tm:                         *時間の入力
        lea     tm_msg,a1
        bsr     disp
        move.w  #$32,d3
        bsr     i_num
        move.b  #$39,d3
        cmp.b   #2,d5
        beq     l3
        bra     l2
l3:     move.b  #$33,d3
l2:     bsr     l1
        move.b  d4,hour
        move.w  #$3a,-(sp)
        DOS     _PUTCHAR
        addq.l  #2,sp
        move.b  #$35,d3
        bsr     i_num
        move.b  #$39,d3
        bsr     l1
        move.b  d4,min
        lea     crlf,a1
        bsr     disp
        rts
l1:
        mulu    #10,d5
        move.b  d5,d4
        bsr     i_num
        add.b   d5,d4
        rts
i_num:
        DOS     _GETC
        cmp.b   #$30,d0
        blt     i_num
        cmp.b   d3,d0
        bhi     i_num
        move.b  d0,d5
        sub.b   #$30,d5
        move.w  d0,-(sp)
        DOS     _PUTCHAR
        addq.l  #2,sp
        rts
com_w:
        move.l  #2,d0           *コモン領域にファイルハンドルを書き込む
        bsr     com_rw
        tst     d0
```

```
        bmi     exit_ok
        rts
thred_open:                     *スレッドを生成する
        clr.l   -(sp)
        pea     mes_buff
        pea     startp
        move.w  #0,-(sp)
        pea     ssp1
        pea     usp1
        move.w  #$10,-(sp)
        pea     thd_name
        DOS     _OPEN_PR
        lea     28(sp),sp
        tst     d0
        bmi     exit_ok
        rts
keep:
        move.l  #endp,a0        *スレッドを常駐終了させる
        sub.l   #startp,a0
        move.l  a0,-(sp)
        DOS     _KEEPPR
*
exit_ok:
        DOS     _EXIT
open_err:
        lea     or_msg,a1
        bsr     disp
        bra     exit_ok
disp:   move.l  a1,-(sp)
        DOS     _PRINT
        addq.l  #4,sp
        rts
        .data
crlf:   dc.b    $0d,$0a,0
thd_name:       dc.b    'NEWZ',0
pw_msg: dc.b    'パスワードを入力して下さい (XXXX;X:半角数字) >',0
tm_msg: dc.b    '時間を入力して下さい(HH:MM ; H:時,M:分)        >',0
start_thd:      dc.b    ' のアクセスを禁止します',$0d,$0a,0
or_msg: dc.b    'そのファイルはオープンできません',$0d,$0a,0
        .stack
        ds.l    256
userp:
        .end    main            * main から実行
```

付　　録

メモリマップ
ディスクマップ
システムファイル
画面構成
SRAMの内容
フロッピディスクコントローラ
IOCS. X

1 メモリマップ

システム関係のメモリマップは、次のようになります。

$000000…… 割り込みベクタ（カッコ内の16進数はベクタ番号）

$00002C	($0B)F系未実装命令トラップベクタ
$00007C	($1F)NMI割り込みベクタ
$0000A0	($28)TRAP 8(システム予約)
$0000A4	($29)TRAP 9(Human68k用DB.Xのブレークポイント)
$0000A8	($2A)TRAP10(POWER OFFまたはリセット処理)
$0000AC	($2B)TRAP11(BREAKキーによるHDOFF処理、そのほか)
$0000B0	($2C)TRAP12(COPYキーによるHCOPY処理、そのほか)
$0000B4	($2D)TRAP13(CTRL+Cキーによるブレークチェックフラグのセット)
$0000B8	($2E)TRAP14(エラー表示と、中止／再実行／無視の選択処理)
$0000BC	($2F)TRAP15(IOCSコール処理)
$000100	($40)MFP(RTCアラーム／1Hz)
$000104	($41)MFP(外部パワーOFF)
$000108	($42)MFP(フロントスイッチOFF)
$00010C	($43)MFP(FM音源)
$000110	($44)MFP(タイマーD、Human68k ver. 2. 0よりPROCESSで使用)
$000114	($45)MFP(タイマーC、マウスRTS／カーソル／FDパワー／他)
$000118	($46)MFP(V-DISP)
$00011C	($47)MFP(RTCクロック)
$000120	($48)MFP(タイマーB)
$000124	($49)MFP(キーシリアル出力エラー)
$000128	($4A)MFP(キーシリアル出力空)
$00012C	($4B)MFP(キーシリアル入力エラー)
$000130	($4C)MFP(キーシリアル入力あり)
$000134	($4D)MFP(タイマーA)
$000138	($4E)MFP(CRTC・IRQ)
$00013C	($4F)MFP(H-SYNC)
$000140	($50)SCCB
$000144	($51)SCCB
$000148	($52)SCCB
$00014C	($53)SCCB
$000150	($54)SCCB(マウス1バイト入力)
$000154	($55)SCCB(マウス1バイト入力)
$000158	($56)SCCB
$00015C	($57)SCCB

1 メモリマップ

	$000160	($58)SCCA
	$000164	($59)SCCA
	$000168	($5A)SCCA
	$00016C	($5B)SCCA
	$000170	($5C)SCCA(RS-232C1 バイト入力)
	$000174	($5D)SCCA(RS-232C1 バイト入力)
	$000178	($5E)SCCA
	$00017C	($5F)SCCA
	$000180	($60)FDCINT(2HD ディスクのステータス割り込み)
	$000184	($61)FDDINT(2HD ディスクの挿入／イジェクト割り込み)
	$000188	($62)HDCINT(ハードディスクのステータス割り込み)
	$00018C	($63)PRNINT(プリンタのレディー割り込み)
	$000190	($64)DMA-A 終わり
	$000194	($65)DMA-A エラー
	$000198	($66)DMA-B 終わり
	$00019C	($67)DMA-B エラー
	$0001A0	($68)DMA-C 終わり
	$0001A4	($69)DMA-C エラー
	$0001A8	($6A)DMA-D 終わり
	$0001AC	($6B)DMA-D エラー
$000400……	($100～$1FF)IOCS コールベクタ	
$000800……	IOCS ワーク	
$001000……	ROM モニタワーク	
$001800……	($FF00～$FFFF)DOS コールベクタ	
$001C00……	Human68k ワーク	
$002000……	IPL 0 トラック 1 セクタがブートされるアドレス(1024 バイト)	
	Human68k のスタック(SSP)	
$006800……	BIOS＋Human68k	
$??????……	Human68k バッファ(files=??、buffers=??、外字エリア)	
$??????……	ユーザーのデバイスドライバ(ここまではスーパーバイザエリア)	
$??????……	シェル(ここから、ユーザーエリア)	
$??????……	ユーザープログラム	
$??????……	フリーエリア	
	$??????	メインメモリ RAMDISK エリア
	$100000	(メモリが入っていない場合は、使用不可)
$C00000……	OS メモリ管理外のエリア(VRAM. I ／ O. ROM)	
	(スーパーバイザエリア)	

2 ディスクマップ

ディスクは次のようにマッピングされ、管理されています。

2.1 2HD ディスク

セクタ番号(1 セクタ 1024 バイトとして)

セクタ番号	
$0000	① 2HD IPL プログラム用
$0001	② 第 1FAT
$0003	第 2FAT(未使用)
$0005	③ ルートディレクトリ
$000B	データエリア
$04CF	

※セクタの番号は、サイド 0 の 0 トラック 1 セクタ目を 0 として、

サイド 0	0 トラック	1 セクタ目	……0
		⋮	⋮
	0 トラック	8 セクタ目	……7
サイド 1	0 トラック	1 セクタ目	……8
		⋮	⋮
	0 トラック	8 セクタ目	……15
サイド 0	1 トラック	1 セクタ目	……16
		⋮	

2.2 ハードディスク：領域を 10 メガバイト確保した場合

セクタ番号(1 セクタ 256 バイトとして)

セクタ番号	
$000000	④ HD IPL メニュープログラム用
$000004	⑤ 管理情報用システム領域
$000005	空きエリア
$000021 ($0000)	⑥ HDIPL プログラム用
$000025 ($0001)	② 第 1FAT
$000075 ($0015)	第 2FAT(未使用)
$0000C5	

2　ディスクマップ

($0029)	③　ルートディレクトリ
$000105	
($0039)	データエリア
$009F51	
($27CC)	⑦　管理外エリア（代替えトラック用）
$009FD7	

↑カッコ内は、OSでのセクタ番号(1セクタ1024バイトとして)

表の丸数字①～⑦には、次のような意味があります。

①2HD IPLプログラム用

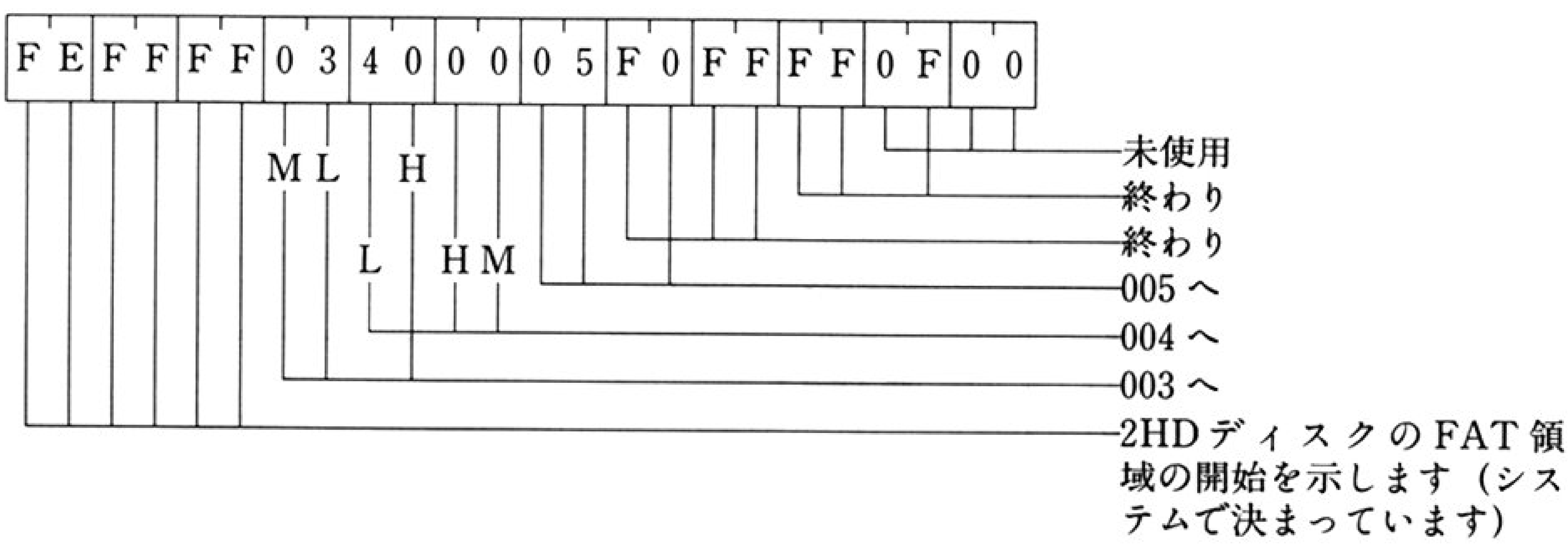
このプログラム自体は$2000番地から動くプログラムであり、起動したドライブの5レコード目（ルートディレクトリの先頭32個分)から、HUMAN. SYSを捜し、それが". X"の形式で作られた$6800番地から動く実行ファイルであれば、それをロードして実行します。
ただし、このファイルは連続セクタにコピーされていなければなりません。
このIPLプログラムはディスクフォーマット時に書き込まれます。

②第1FAT

FAT(File Allocation Table)とは、ファイルのディスク上のつながりを管理するもので、ファイル1024バイトに対し1個のFATが使われます。
ただし、FATが先に確保される関係で、0バイトのファイルでも一個のFATが使われます。
以後1024バイト単位で1個づつ増える計算になります。
FAT領域の先頭2個分はシステムで決められており、2HDの場合は$FEFFFFの3バイト、ハードディスクの場合は領域確保の容量により4メガバイトまでは$F8FFFFの3バイト、5メガバイト以上は$F8FFFFFFの4バイトになります。
FATには1.5バイトFATと2バイトFATがあり、4メガバイトまでは1.5バイト、5メガバイトから2バイトFATとなります。

●1.5バイトFATの構造(3バイトで2個のFATを表わします)

(High、Middle、Lowの順に読む)



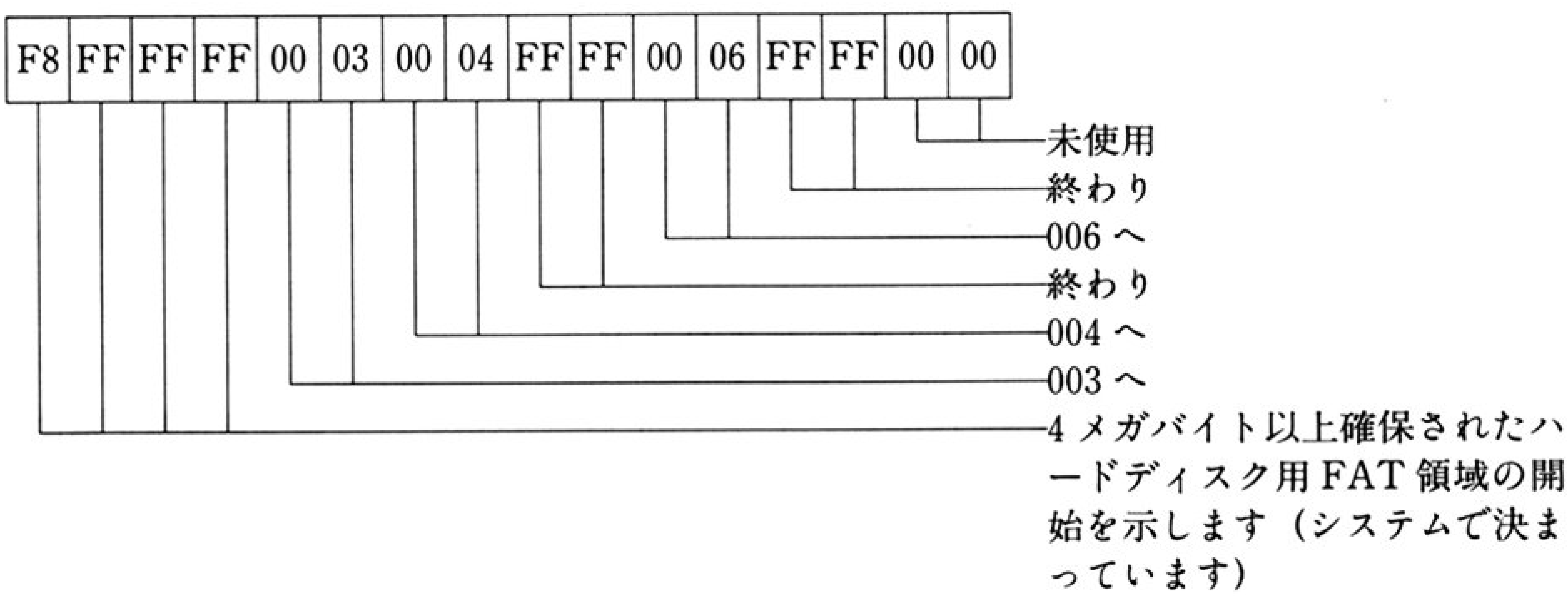
2 ディスクマップ

● FAT の値の意味(1.5バイト FAT)

FAT の値	意　　味
$000	未使用エリアであることを表します
$002～$FF6	次の FAT 番号を表します
$FF7	使用不能クラスタであることを表します
$FFF	このクラスタで終わりであることを表します

※クラスタとは、X68000 でディスクを読み書きする単位のことです。
通常 1 セクタ＝1 クラスタです。

●2バイト FAT の構造(2バイトで一個の FAT を表わします)



● FAT の値の意味(2バイト FAT)

FAT の値	意　　味
$0000	未使用エリアであることを表します
$0002～$FFF6	次の FAT 番号を表します
$FFF7	使用不能クラスタであることを表します
$FFFF	このクラクタで終わりであることを表します

2 ディスクマップ

③ルートディレクトリ

ディレクトリは一つあたり32バイトからなり、2HDディスクには192個、ハードディスクには512個まで作ることができます。
ディレクトリエリアには、通常ファイルのほかに、ボリューム名や、サブディレクトリ名も入ります。
32バイトの内容は次の通りです。

$0000

(8バイト)ファイル名1
先頭1文字が$E5なら消されたファイル、$00ならディクトリの終わりです。
$E5で始まるファイル名は$05とします。
残りはスペース($20)で埋めてあります。

$0008

(3バイト)拡張子
残りはスペース($20)で埋めてあります。

$000B

(1バイト)アトリビュート

MSB	D07	D06	D05	D04	D03	D02	D01	D00	LSB
	0	0	A	D	V	S	H	R	

- R — 書き込み禁止属性
- H — 不可視属性
- S — システム属性
- V — ボリューム名
- D — ディレクトリ名
- A — 通常のファイル

$000C

(10バイト)8バイト以上のファイル名の残り
残りは$00で埋めてあります。

$0016

(2バイト)変更最終時刻

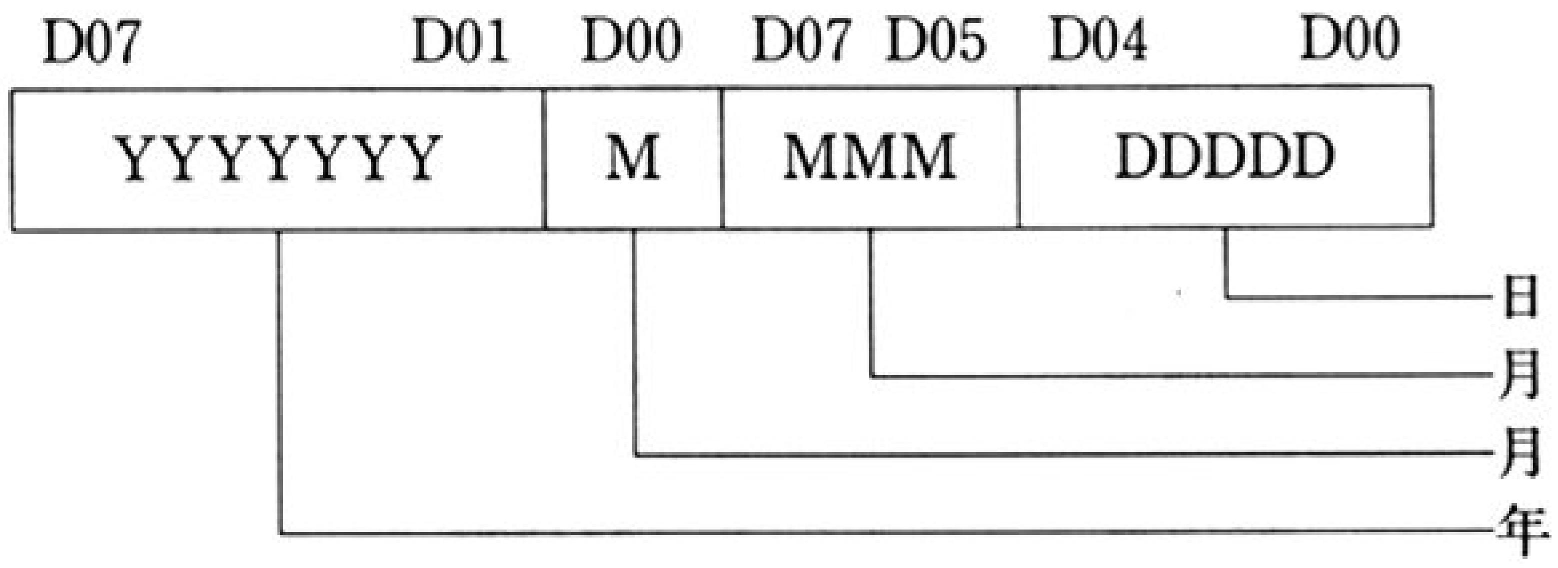
MSB	D07 — D03	D02 — D00	D07 — D05	D04 — D00	LSB
	HHHHH	MMM	MMM	SSSSS	

- SSSSS — 秒×2
- MMM (D07–D05) — 分
- MMM (D02–D00) — 分
- HHHHH — 時

バイトの並びは、MMMSSSSSの後、HHHHHMMMです。

$0018

(2バイト)変更最終月日



バイトの並びは、MMMDDDDDの後、YYYYYYYMです。

$001A

(2バイト)最初のFAT番号
バイトの並びは、下位8ビット、上位8ビット。

$001C

(4バイト)ファイルサイズ
バイトの並びは、下位8ビットから、上位へ。

$0020

④ HD IPL メニュープログラム用

このプログラム自体は$2000番地から動作し、HDをいくつかのシステムで使用する場合、どのシステムを起動するかを聞いてくるプログラムです。
単体のシステムですべての領域を確保している場合は、単に⑥のHD IPLプログラムを$2400番地にロードして起動するだけです。
このプログラムはハードディスクフォーマット時に書き込まれます。

⑤管理情報用システム領域

HD上に複数のシステムを確保する場合の情報を書き込みます。
管理情報は16バイトずつ16のブロックに分けられます。
先頭の1ブロックは、ドライブの情報、残りの15ブロックにはシステムの情報が書かれます。

(1) 先頭のドライブ情報ブロック

1.	'X68K'	4バイト
2.	最大レコード番号	4バイト
3.	代替トラックの先頭のレコード番号	4バイト
4.	シッピングゾーンのレコード番号	4バイト

(2) その他の15ブロック

1.	システム名	8バイト
2.	先頭レコード番号	4バイト
3.	レコード長	4バイト

2　ディスクマップ

(参考)ドライブ情報ブロックの値

容　量	最大レコード	代替えレコード	HDOFF レコード
10 メガバイト	$9F54	$9F54	$AF50
20 メガバイト	$139C8	$139C8	$15660
40 メガバイト	$27930	$27930	$2ACC0

先頭レコードの最上位 8 ビットが 0 でない場合は、現在無効な領域となります。
データは FORMAT の初期化で書き込まれます。
先頭の 16 バイト以外は、FORMAT の領域確保、領域選択、領域開放で書き換えられます。

⑥ HD IPL プログラム用

このプログラム自体は$2400 番地から動作し、起動したドライブのルートディレクトリの先頭 32 個の中から HUMAN. SYS を捜し、それが“. X”の形式で作られた$6800 番地から動く実行ファイルであれば、それをロードして実行します。
ただしこのファイルは連続セクタにコピーされていなければなりません。

⑦管理外エリア(代替えトラック用)

通常は使用されません。
不良トラックの代替え用です。

3 システムファイル

Human68k は、拡張子が“. SYS”であるようなファイルをシステム専用ファイルとして扱い、それらによって色々な機能を提供しています。
システム専用ファイルはおもに Human68k 起動時に読み込まれ、次に示すようなものがあります。

タイトル表示ファイル(TITLE. SYS)
外字ファイル(USKCG. SYS)
キー定義ファイル(KEY. SYS)
ビープ音声ファイル(BEEP. SYS)

3.1 タイトル表示ファイル

Human68k は、起動時に TITLE. SYS というファイルがあると、標準のタイトルを出さずに、TITLE. SYS に従ったタイトルを表示します。
TITLE. SYS は$10044 バイト以下のファイルで、テキストタイトルとビットイメージタイトルの2種類のタイプがあります。
テキストタイトルはコマンドモード用に用意されたもので、タイトル表示ファイルの文字列を表示します。
ビットイメージタイトルはビジュアルモード用のタイトル表示ファイルで、ビットイメージデータを表示します。

1. テキストタイトル(先頭1バイトが0でないこと)
   タイトル表示ファイル中の文字列を、画面に表示します。
   文字列の終了コードは0です。
   コントロールコード、エスケープシーケンスも使用可能です。
2. ビットイメージタイトル（先頭1バイトが0であること）

3　システムファイル

以下にその構造を示します

ファイル先頭からのオフセット	サイズ(バイト)	意　　味
+0	2	$0000 にしておきます。
+2	1	表示するテキスト画面の指定 1 ＝ テキスト 0 2 ＝ テキスト 1 4 ＝ テキスト 2 8 ＝ テキスト 3
+3	1	塗りつぶすテキスト画面の指定 0 ＝ 塗りつぶす面はない 1 ＝ テキスト 0 2 ＝ テキスト 1 4 ＝ テキスト 2 8 ＝ テキスト 3
+4	2	テキスト 2 のパレット番号
+6	2	テキスト 3 のパレット番号
+8	$34	シエルの指定(“SHELL＝”の右辺)，0
+60	2	表示する X 座標の先頭
+62	2	表示する Y 座標の先頭
+64	2	ビットイメージデータの X 方向のドット数
+66	2	ビットイメージデータの Y 方向のドット数
+68	?	ビットイメージデータ

3.2 外字ファイル

Human68k は起動時に、USKCG. SYS というファイルがあると、外字ファイルとして、Human68k 内の外字エリアに読み込みます。
USKCG. SYS は$10000バイト以下のファイルで、その作成、修正は USKCGM. X で行います。
全角の外字コードは JIS 漢字コード、シフト JIS コードのどちらでもかまいませんが、JIS 漢字コードは LOW／HIGH が逆になっているので、注意してください。

ファイル先頭からのオフセット	サイズ(バイト)	意　味
+0	34	未使用
+34	2	外字 JIS 漢字コード
+36	32	フォントデータ
	2	外字シフト JIS コード
	32	フォントデータ
	2	半角外字シフト JIS コード
	16	フォントデータ
		:
		:
	2	16×16 から 24×24 への切り換えコード($FFFF)
	2	外字 JIS 漢字コード
	72	フォントデータ
	2	外字シフト JIS コード
	72	フォントデータ
	2	半角外字シフト JIS コード
	48	フォントデータ
		:
		:

3.3 キー定義ファイル

Human68k は起動時に、KEY. SYS というファイルがあると、ファンクションキーのデータとして定義します。

KEY. SYS は 712 バイトのファイルで、KEY. X によって作成、変更が行えます。

先頭が$FE で始まるキーは、$FE 以後 8 文字は表示用として使われ、9 文字目からが本当のキーコードとなります。

文字列の終わりは、必ず 0 で終わらなければなりません。

ファイル先頭からのオフセット	サイズ(バイト)	意　味
+0	32	F1 のデータ
+32	32	F2 のデータ
+64	32	F3 のデータ
+96	32	F4 のデータ
+128	32	F5 のデータ
+160	32	F6 のデータ
+192	32	F7 のデータ
+224	32	F8 のデータ
+256	32	F9 のデータ
+288	32	F10 のデータ
+320	32	SHIFT+F1 のデータ
+352	32	SHIFT+F2 のデータ
+384	32	SHIFT+F3 のデータ
+416	32	SHIFT+F4 のデータ
+448	32	SHIFT+F5 のデータ
+480	32	SHIFT+F6 のデータ
+512	32	SHIFT+F7 のデータ
+544	32	SHIFT+F8 のデータ
+576	32	SHIFT+F9 のデータ
+608	32	SHIFT+F10 のデータ
+640	6	ROLL UP
+646	6	ROLL DOWN
+652	6	INS
+658	6	DEL

+664	6	上向きカーソルキー
+670	6	左向きカーソルキー
+676	6	右向きカーソルキー
+682	6	下向きカーソルキー
+688	6	CLR
+694	6	HELP
+700	6	HOME
+706	6	UNDO

3.4 ビープ音声ファイル

Human68k は CONFIG. SYS の中に、“BELL=BEEP. SYS” と指定すると、BEEP. SYS を ADPCM のデータファイルとして、Human68k 内に読み込みます。

データはサンプリング周波数 15.6KHz で$FFFF バイト以内の、ADPCM から読み込んだデータです。

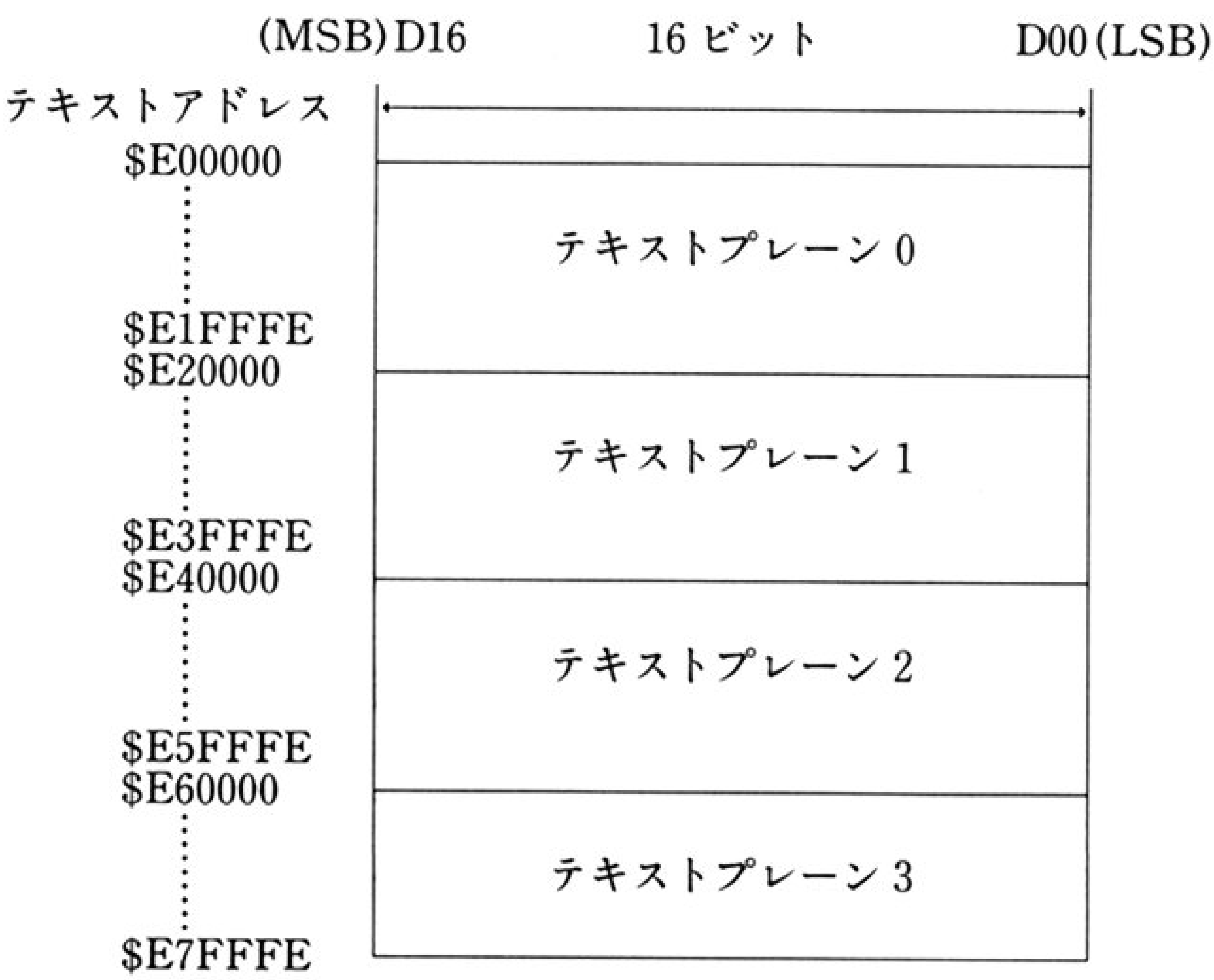
4 画面構成

4.1 テキスト画面構成

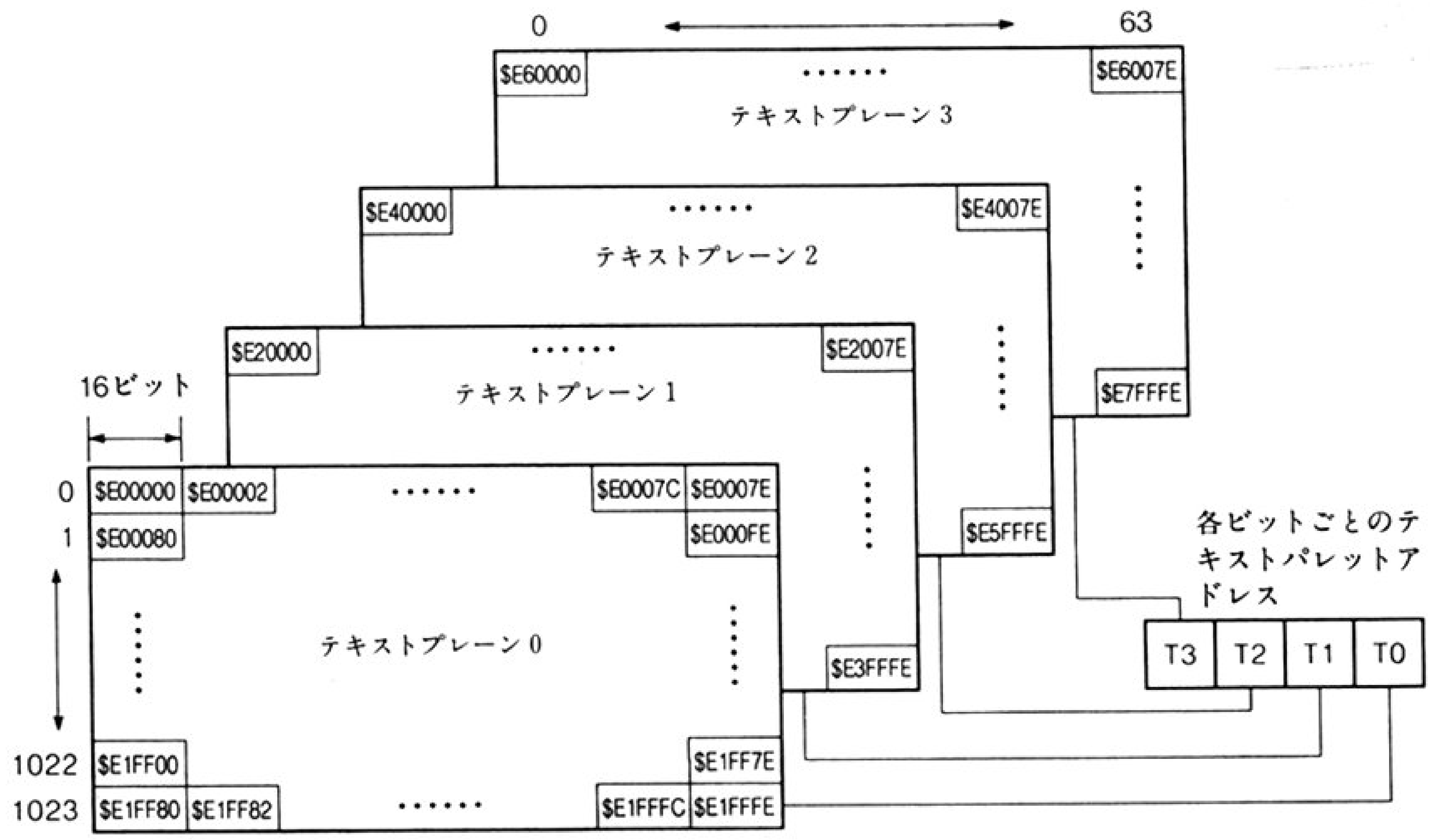
①表示画面構成

実画面 (H×V)	同時表示色 (パレット色)	同時表示面数 (重ね合せ面数)	表示画面 (H×V)	備　　考
1024×1024	16 (65536)	1	高解像度モード 768×512 512×512 512×256 256×256	 2 度読み 2 度読み
			標準解像度モード (オーバースキャン) 512×256 512×512 256×256	・オーバースキャンのみスーパーインポーズ ・オーバースキャンのため実際の表示画面サイズは小さくなる インターレース

②テキスト VRAM のメモリマップ



③テキスト実画面のアドレス配置



④テキストパレットアドレス
[READ/WRITE 可]

パレットアドレス	レジスタアドレス	D15〜D11	D10〜D06	D05〜D01	D00	備　　考
		Green	Red	Blue	I	
$00 $01 $02 $03 ⋮ $0C $0D $0E $0F	$E82200 $E82202 $E82204 $E82206 ⋮ $E82218 $E8221A $E8221C $E8221E	16 ビットデータ				ただし、 このパレットアドレスはスプライトカラー0のパレットと共通に使用するものとする。

※たとえば、テキスト VRAM のメモリデータが

$E00000 = $0000　　$E40000 = $0002
$E20000 = $0002　　$E60000 = $0002

なら、そのときの D01 ビット(水平 14 ビット、垂直 0 ライン目)のパレットアドレスは$0E になります

上表より、そのパレットアドレスは$E8221C ですから、出力されるパレットデータは$E8221C の内容となります。

※パレットアドレス$00 のデータには、原則として$0000 を入れてください。

4　画面構成

4.2 グラフィック画面構成

①表示画面構成

実画面 (H×V)	同時表示色 (パレット色)	同時表示面数 (重ね合せ面数)	表示画面 (H×V)	備　　考
1024×1024	16 (65536)	1	高解像度モード	
			768×512	
			512×512	
			512×256	2度読み
			256×256	2度読み
			標準解像度モード (オーバースキャン)	・オーバースキャンのみスーパーインポーズ ・オーバースキャンのため実際の表示画面サイズは小さくなる
			512×512	インターレース
			512×256	
			256×256	
512×512	16 (65536)	4	高解像度モード	
			512×512	
			512×256	2度読み
			256×256	2度読み
	256 (65536)	2	標準解像度モード (オーバースキャン)	・オーバースキャンのみスーパーインポーズ ・オーバースキャンのため実際の表示画面サイズは小さくなる
	65536 (65536)	1	512×512	インターレース
			512×256	
			256×256	

②グラフィック VRAM のメモリマップ

〈1〉 実画面1024×1024、16色、1面モード（＊は無効）

MSB　LSB

(アドレス)	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00	
$C00000	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X0, Y0)
$C00002	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X1, Y0)
⋮						⋮								⋮			⋮
$DFFFFE	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X1023,Y1023)

（D03～D00：グラフィックパレットアドレス）

〈2〉 実画面512×512、16色、4色モード（＊は無効）

MSB　LSB

(アドレス)	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00	
$C00000	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X0, Y0)
PG0 (ページ 0)						⋮								⋮			⋮
$C7FFFE	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X511, Y511)
$C80000	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X0, Y0)
PG1 (ページ 1)						⋮								⋮			⋮
$CFFFFE	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X511, Y511)
$D00000	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X0, Y0)
PG2 (ページ 2)						⋮								⋮			⋮
$D7FFFE	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X511, Y511)
$D80000	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X0, Y0)
PG3 (ページ 3)						⋮								⋮			⋮
$DFFFFE	*	*	*	*	*	*	*	*	*	*	*	*	G03	G02	G01	G00	(X511, Y511)

（D03～D00：グラフィックパレットアドレス）

4　画面構成

〈3〉実画面512×512、256色、2面モード(＊は無効)

MSB　　LSB

(アドレス)	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00	
$C00000	＊	＊	＊	＊	＊	＊	＊	＊	G07	G06	G05	G04	G03	G02	G01	G00	(X0, Y0)
PG0 (ページ0)							⋮							⋮			⋮
$C7FFFE	＊	＊	＊	＊	＊	＊	＊	＊	G07	G06	G05	G04	G03	G02	G01	G00	(X511, Y511)
$C80000	＊	＊	＊	＊	＊	＊	＊	＊	G07	G06	G05	G04	G03	G02	G01	G00	(X0, Y0)
PG1 (ページ1)							⋮							⋮			⋮
$CFFFFE	＊	＊	＊	＊	＊	＊	＊	＊	G07	G06	G05	G04	G03	G02	G01	G00	(X511, Y511)

D07〜D00：グラフィックパレットアドレス

〈4〉実画面512×512、65536色、1面モード

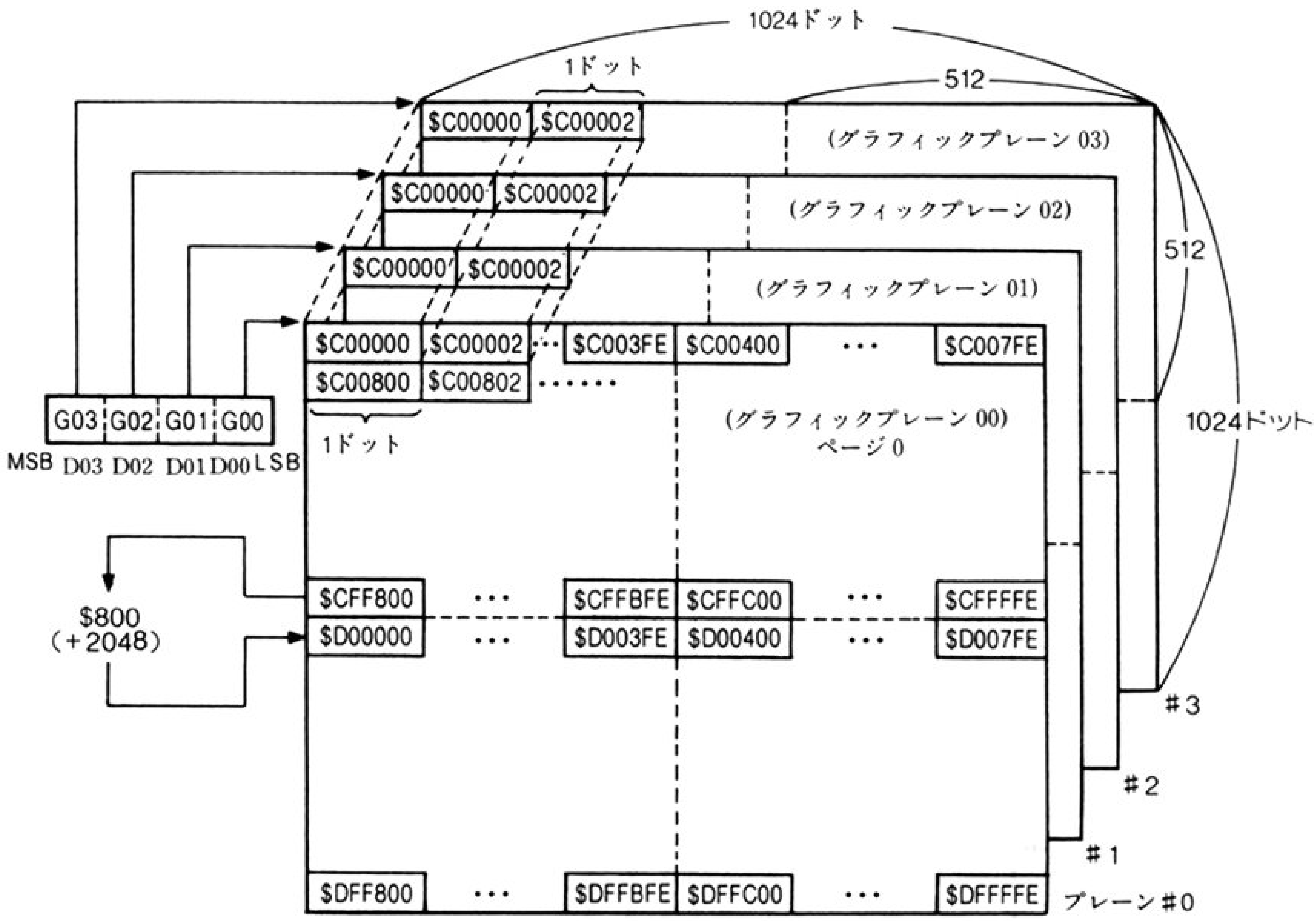
MSB　　LSB

(アドレス)	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00	
$C00000	G15	G14	G13	G12	G11	G10	G09	G08	G07	G06	G05	G04	G03	G02	G01	G00	(X0, Y0)
$C00002	G15	G14	G13	G12	G11	G10	G09	G08	G07	G06	G05	G04	G03	G02	G01	G00	(X1, Y0)
⋮							⋮							⋮			⋮
$C7FFFE	G15	G14	G13	G12	G11	G10	G09	G08	G07	G06	G05	G04	G03	G02	G01	G00	(X511, Y511)

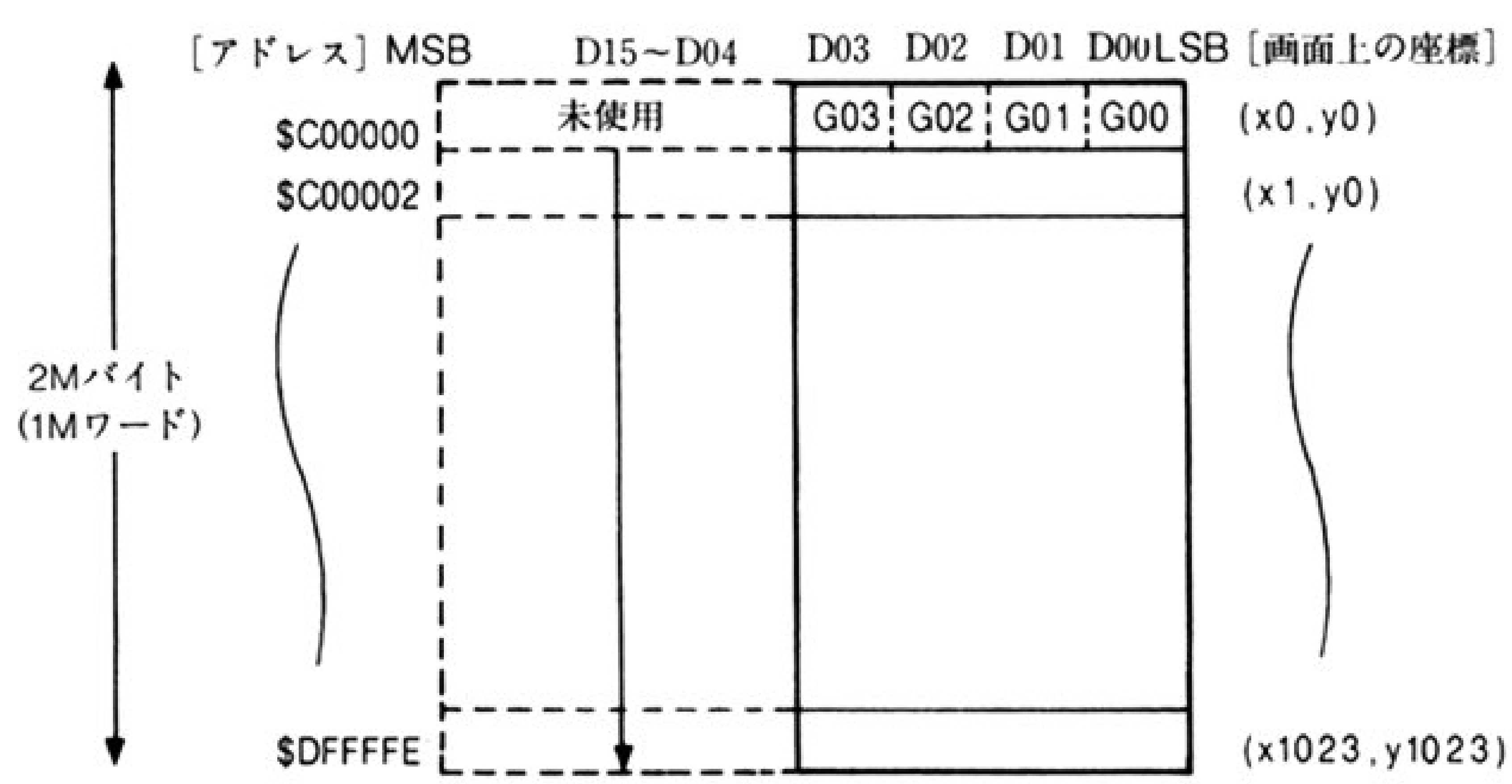
D15〜D00：グラフィックパレットアドレス

③グラフィック実画面のアドレス配置

〈1〉1024×1024　ドット(16色モード)



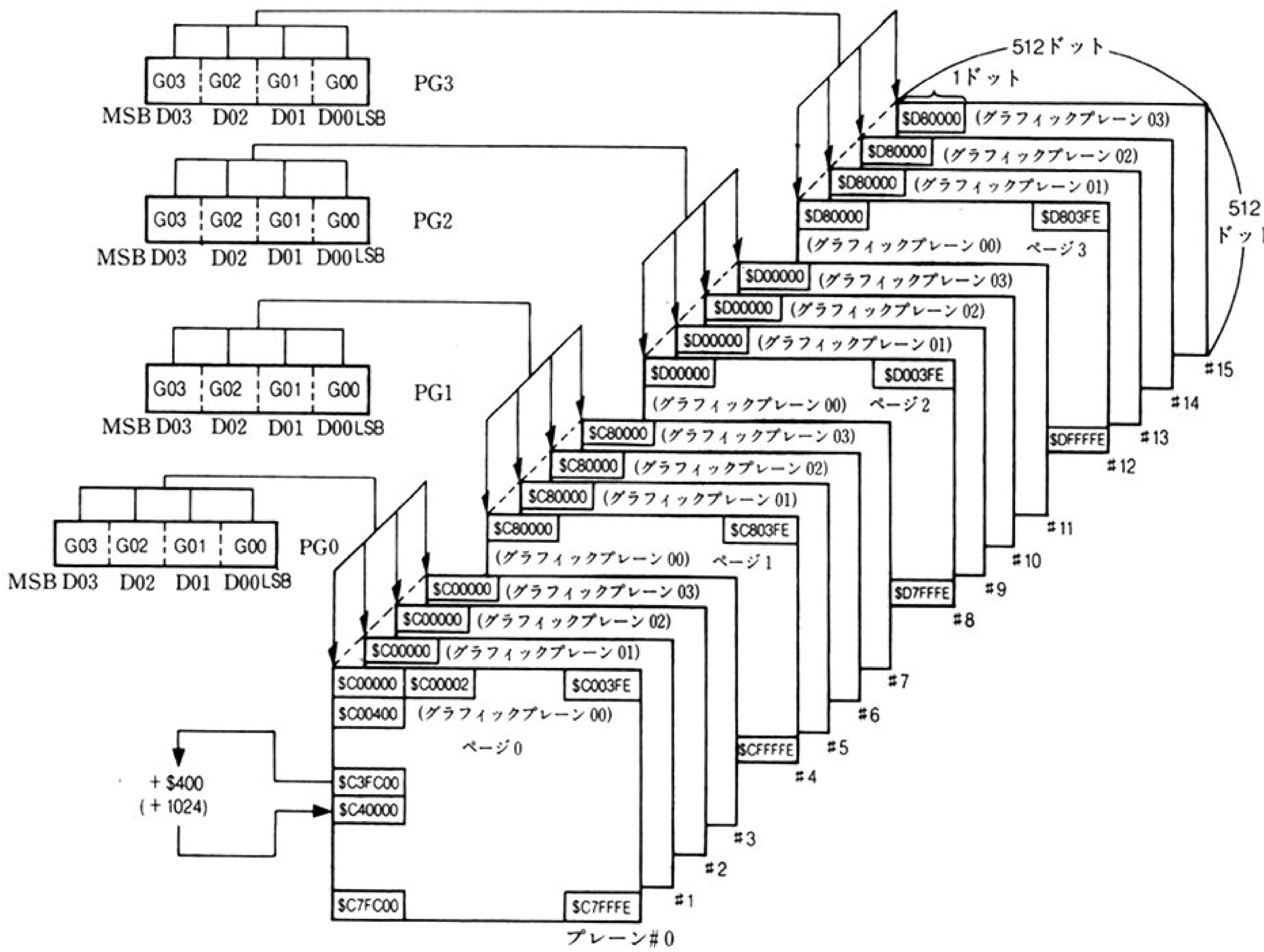
グラフィック仮想画面のアドレス配置 (1)



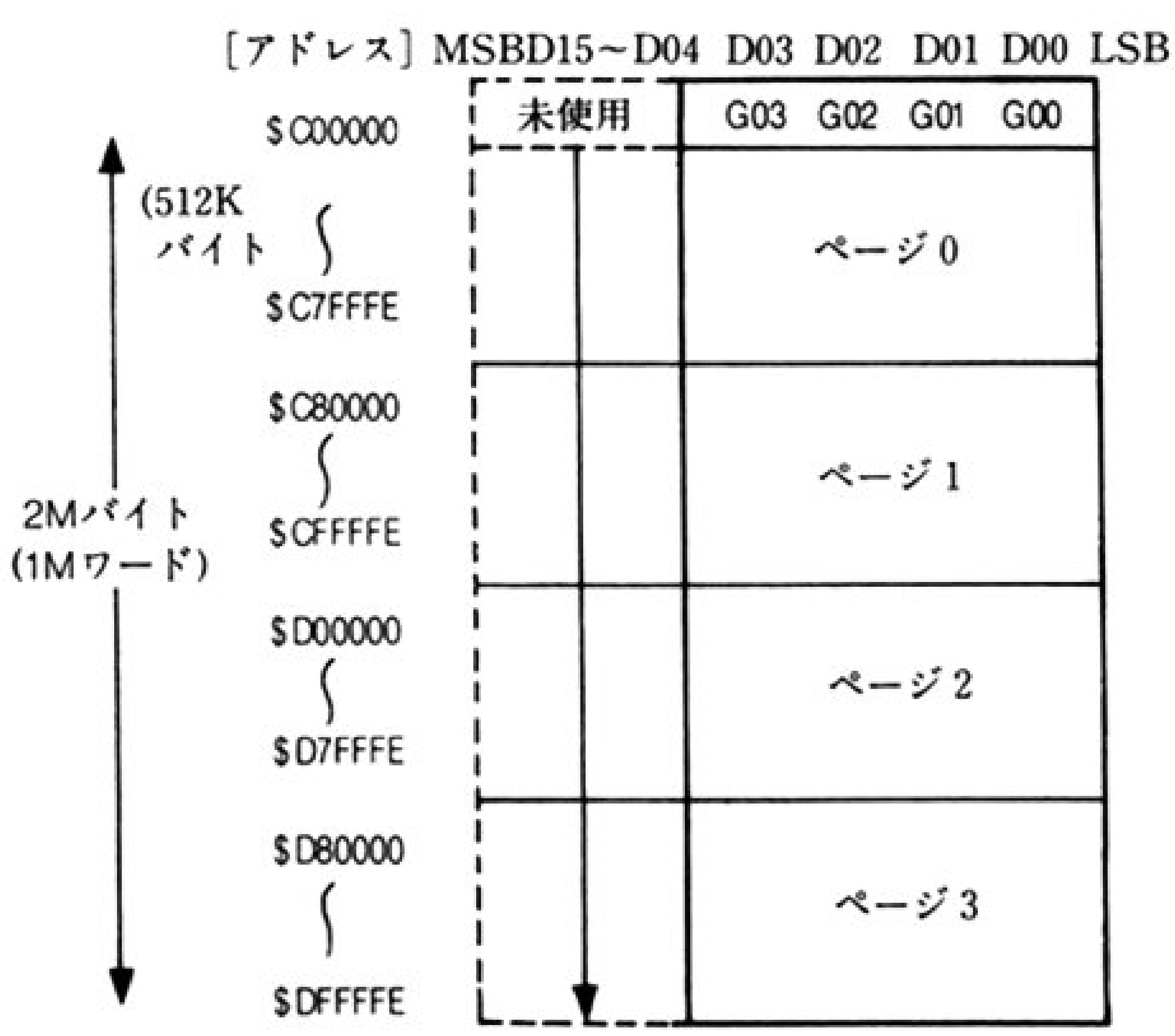
メモリマップ

4　画面構成

〈2〉 512×512ドット(16色、4面モード)



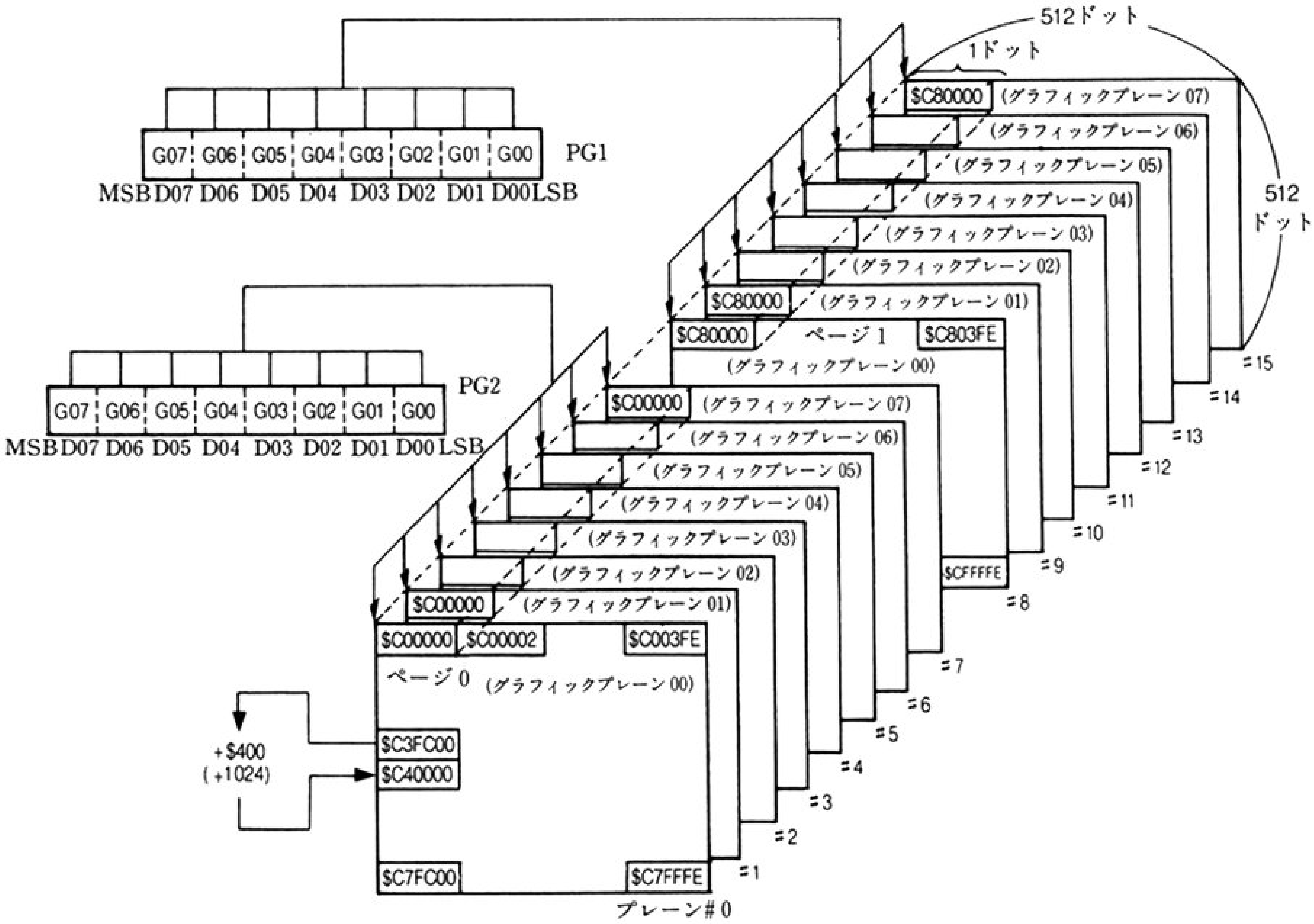
グラフィック仮想画面のアドレス配置 (2)



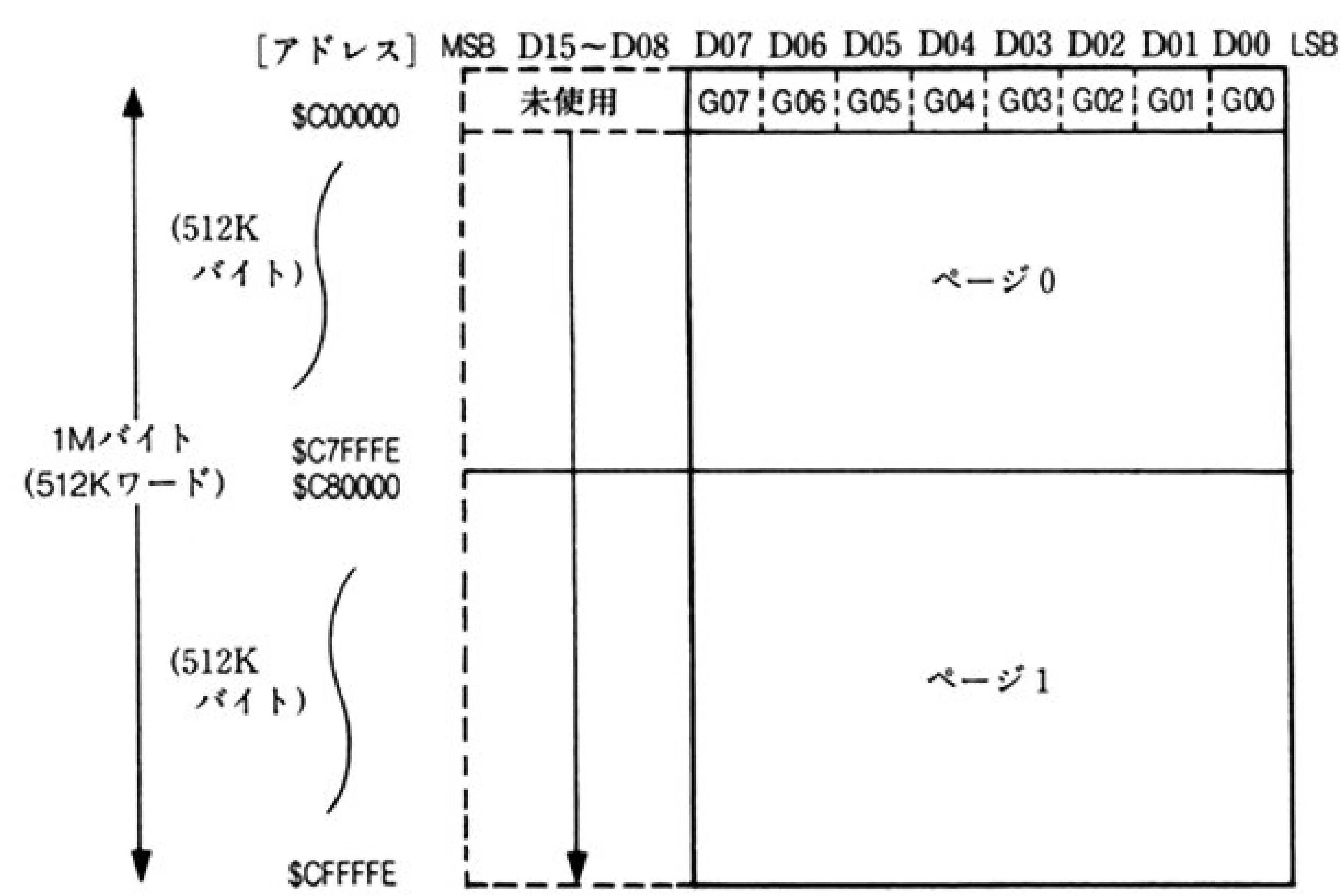
メモリマップ

4 画面構成

〈3〉512×512ドット(256色、2面モード)



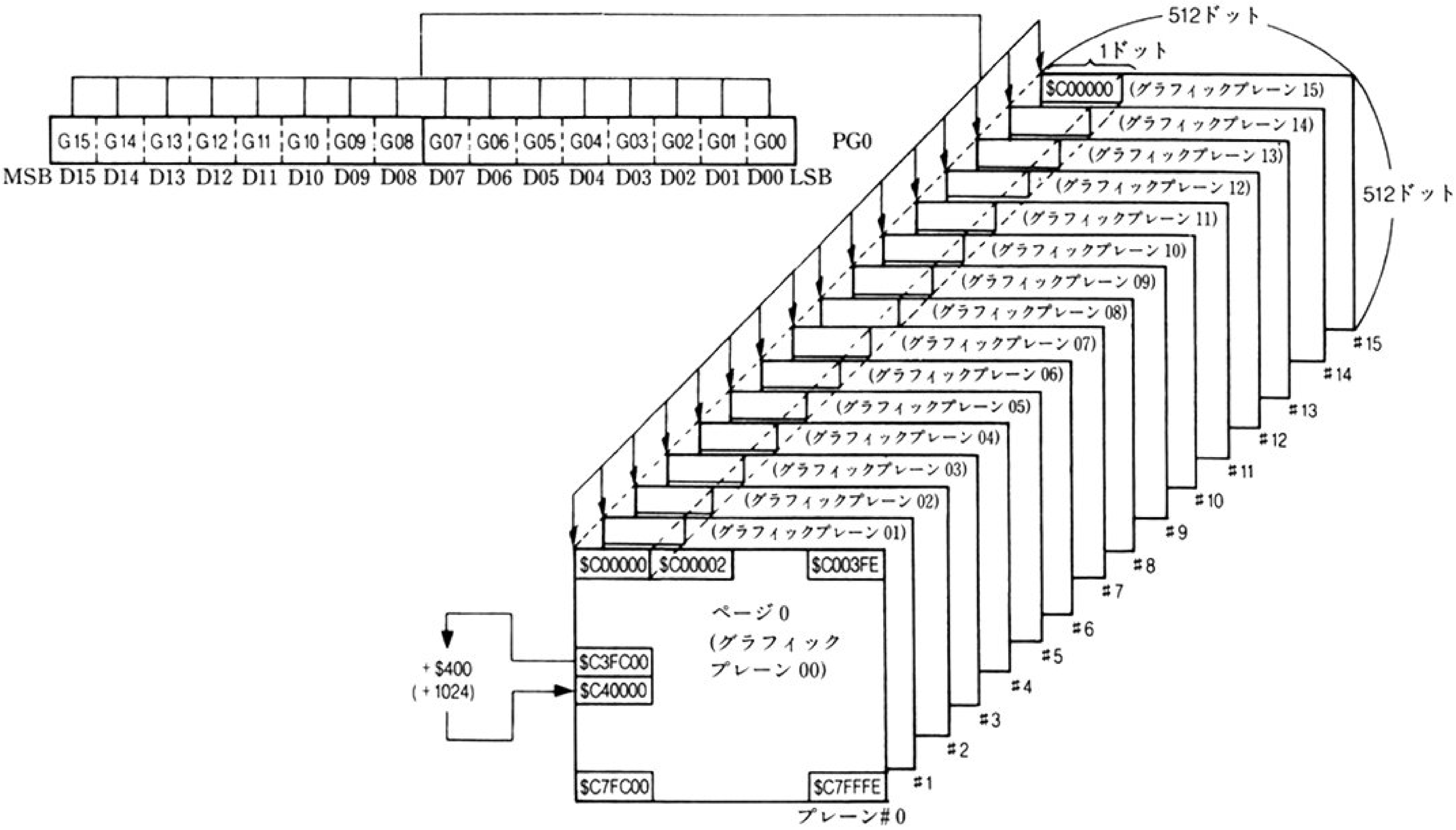
グラフィック仮想画面のアドレス配置(3)



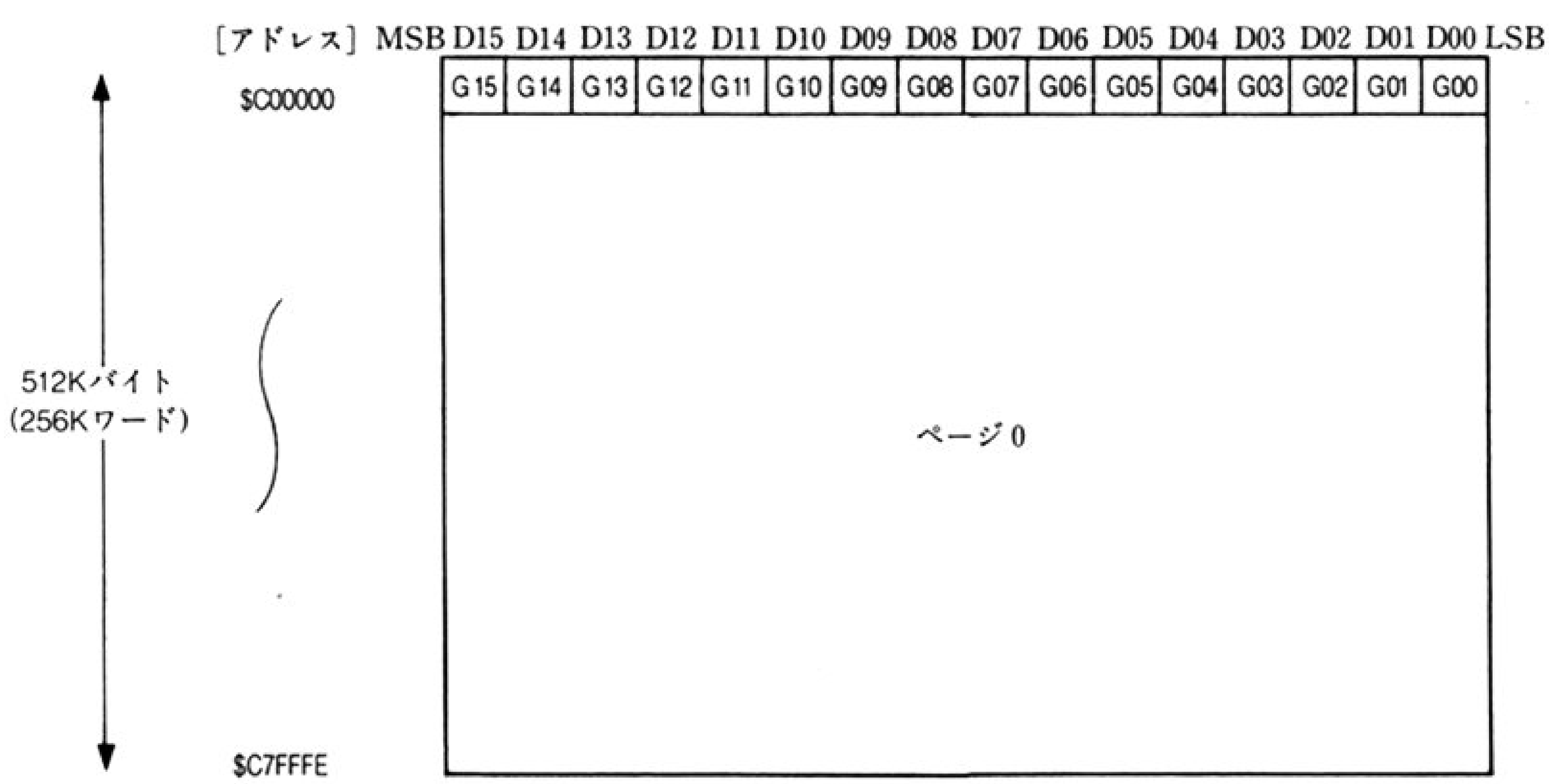
メモリマップ

4　画面構成

〈4〉512×512ドット(65536色、1面モード)



グラフィック仮想画面のアドレス配置(4)



メモリマップ

4　画面構成

④グラフィックパレットアドレス

〈1〉グラフィック16色、256色モード　[READ/WRITE 可]

パレットアドレス VRAMデータ	レジスタアドレス	D15〜D11	D10〜D06	D05〜D01	D00	備　考	
		Green	Red	Blue	I		
$00	$E82000	16ビットデータ				16色モードパレット	256色モードパレット
$01	$E82002						
$02	$E82004						
$03	$E82006						
⋮	⋮						
$0C	$E82018						
$0D	$E8201A						
$0E	$E8201C						
$0F	$E8201E						
$10	$E82020						
$11	$E82022						
$12	$E82024						
⋮	⋮						
$FD	$E821FA						
$FE	$E821FC						
$FF	$E821FE						

※グラフィック256色2面モードのとき、たとえばグラフィックVRAMのメモリデータが$11だったとすると、そのときのパレットレジスタアドレスは上表より$E82022、したがって出力されるパレットデータは$E82022の内容となります。

また、16色の場合もパレットアドレスの数が違うだけで同様です。

※パレットアドレス$00のデータには、原則として$0000を入れてください。

4　画面構成

〈2〉グラフィック65536色モード［READ/WRITE 可］

パレットアドレス グラフィックVRAMメモリデータ下位バイト		レジスタアドレス	D15〜D14	D13〜D09	D08	D07〜D06	D05〜D01	D00	備考
			Red 下位	Blue	I	Red 下位	Blue	I	
$00	$01	$E82000	8ビット			8ビット			VRAMのメモリデータの下位バイトが偶数ならD15〜D8の内容を、奇数ならD7〜D0の内容をパレットデータの下位バイトとして出力します。
$02	$03	$E82004							
$04	$05	$E82008							
⋮	⋮	⋮							
$FC	$FD	$E821F8							
$FE	$FF	$E821FC							

パレットアドレス グラフィックVRAMメモリデータ上位バイト		レジスタアドレス	D15〜D11	D10〜D08	D07〜D03	D02〜D00	備考
			Green	Red 上位	Green	Red 上位	
$00	$01	$E82002	8ビット		8ビット		VRAMのメモリデータの上位バイトが偶数ならD15〜D8の内容を、奇数ならD7〜D0の内容をパレットデータの上位バイトとして出力します。
$02	$03	$E82006					
$04	$05	$E8200A					
⋮	⋮	⋮					
$FC	$FD	$E821FA					
$FE	$FF	$E821FE					

※それぞれのパレットレジスタアドレスのデータには、原則としてそのパレットアドレス値を入れてください。

特に、グラフィック65536色1面モードのときのパレット構成は上表のようになっており、上、下位1ビットを変えるだけで対応する256色のパレットが変わります。

あらかじめデータを設定するときには、16色や256色モードとは異なることに注意してください。

※グラフィック65536色1面モードのとき、たとえばグラフィックVRAMのメモリデータが$0101だとすると、そのときのパレットレジスタアドレスは上表より$E82002と$E82000となり、出力されるパレットデータは$E82002のD07〜D00と$E82000のD07〜D00を組み合わせた値となります。

すなわち、$E82000が$3521で、$E82002が$4CFFなら、パレット出力データは$FF21になります。

5 SRAMの内容

X68000は、バッテリバックアップされたスタティックRAM(SRAM)を16Kバイト持っています。

このメモリは$ED0000～$ED3FFFまでにあり、読み込むことはできますが、書き込むためには、次の手続きが必要です。

```
MOVE. B    #$31, $E8E00D          書き込み許可
                   ⋮
(この間で書き込み処理をしてください)
                   ⋮
MOVE. B    #0, $E8E00D            書き込み禁止
```

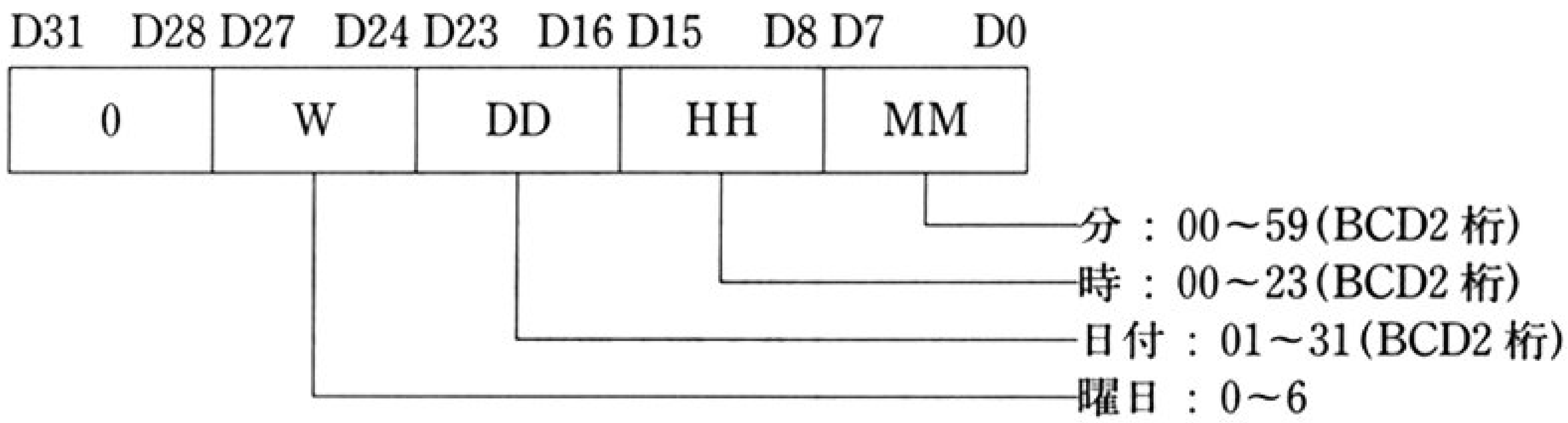
SRAMのメモリマップと内容は次の通りです。

アドレス	サイズ、初期値	意　　　　味
$ED0000	8 'X 68000',$57 全角 半角 (2 +5→7バイト)	メモリチェックデータ(このデータが壊れた場合は、SRAMは初期化されます。
$ED0008	4 $100000	メモリのリミットアドレス+1が入るワーク
$ED000C	4 $BFFFFC	ROMブートのスタートアドレスが入っているポインタのワーク
$ED0010	4 $ED0100	SRAMブートのスタートアドレスのワーク
$ED0014	4 −1	IOCSコールALARMSET($5E)のD2-1のワーク。 アラームで起動したあとOFFするまでの時間(分単位)です。 −1なら無限大を表します。
$ED0018	2 0	最優先にブートするデバイスのタイプを指定します。 =$0000 STD 2HD -HD -ROM -RAM =$8?00 HD? HD -STD =$9?70 2HD? 2HD -STD =$A000 ROM ROM -STD =$B000 RAM RAM -STD ?は、ドライブ番号です。 最優先のデバイスが準備できていないときはSTDとなります。
$ED001A	2 $4E07	RS-232Cの設定(次図参照)。

5　SRAMの内容

アドレス	サイズ、初期値	意　　味
		D15 D14 D13 D12 D11 D10 D09 D08 D07 D06 D05 D04 D03 D02 D01 D00 ST(D15–D14) PP(D13–D12) BB(D11–D10) ×(D09) ×(D08) ×(D07) ×(D06) ×(D05) ×(D04) ×(D03) BAUD(D02–D00) ストップビット 01 : 1 10 : 1.5 00, 11 : 2 パリティ 01 : 奇数 11 : 偶数 ?0 : パリティなし ビット長 00 : 5 ビット 01 : 6 ビット 10 : 7 ビット 11 : 8 ビット ボーレート指定 000 : 75 001 : 150 010 : 300 011 : 600 100 : 1200 101 : 2400 110 : 4800 111 : 9600
$ED001C	1 0	パワーオン時のキー LED の状態(OFF するときの状態が退避されています)。下図参照。
		D07 D06 D05 D04 D03 D02 D01 D00 D07 : 0 D06 : 全角 D05 : ひらがな D04 : INS D03 : CAPS D02 : コード入力 D01 : ローマ字 D00 : かな 0 : LED 消灯 1 : LED 点灯
$ED001D	1 16	IPL での面画モード(IOCS コール CRTMOD($10) の D1 の値)。IOCS 内のエラー表示ルーチンは 16 のモードにのみ対応していますので、できるだけ 16 以外の値はセットしないでください。 Human68k はつねに 16 のモードで起動します。
$ED001E	4 0	アラームで起動したとき、何をするかのワーク。 IOCS コール ALARMSET($5E)の A1 の値。 0 :普通に立ち上がります(指定のメディアからブートします)。 OFF するまでの時間が−1 でないなら、指定時間後本体を OFF。 1〜$3F :テレビをコントロールします。 OFF するまでの時間が−1 なら 1 分後本体のみ OFF。

5　SRAMの内容

<table>
<tr><th>アドレス</th><th>サイズ、初期値</th><th>意　　　味</th></tr>
<tr><td></td><td></td><td>それ以外なら、一定時間後、本体とテレビをOFF。
$40以上：$40～$FFFFFFFFまでのアドレス指定であり、先頭1バイトが$60ならスーパーバイザのままそのアドレスをコールします。そのサブルーチンから戻ると、本体をOFFします。
−1：ふつうに立ち上がります(指定のメディアからブートします)。
テレビの制御はしないで立ち上がります。
OFFするまでの時間が−1でないなら、指定時間後本体をOFF。</td></tr>
<tr><td>$ED0022</td><td>4　$FFFF0000</td><td>アラームで起動する時間のワーク。
IOCSコールALARMSET($5E)のD1の値。
それぞれ$Fは無視することを表します(下図参照)。</td></tr>
<tr><td colspan="3">

(例)　毎月5日10時50分　=$0F051050
毎週日曜日12時00分=$00FF1200
毎日18時10分　=$0FFF1810</td></tr>
<tr><td>$ED0026</td><td>1　7</td><td>アラームの禁止／許可フラグ。
IOCSコールALARMMOD($5D)やALARMSET($5E)で変化するワーク。
0：許可
7：禁止</td></tr>
<tr><td>$ED0027</td><td>1　0</td><td>OPT.2 キーをテレビコントロールキーとして使用するかどうかのフラグ。
0：テレビコントロールキーとして使用
1：通常のモードキーとして使用</td></tr>
</table>

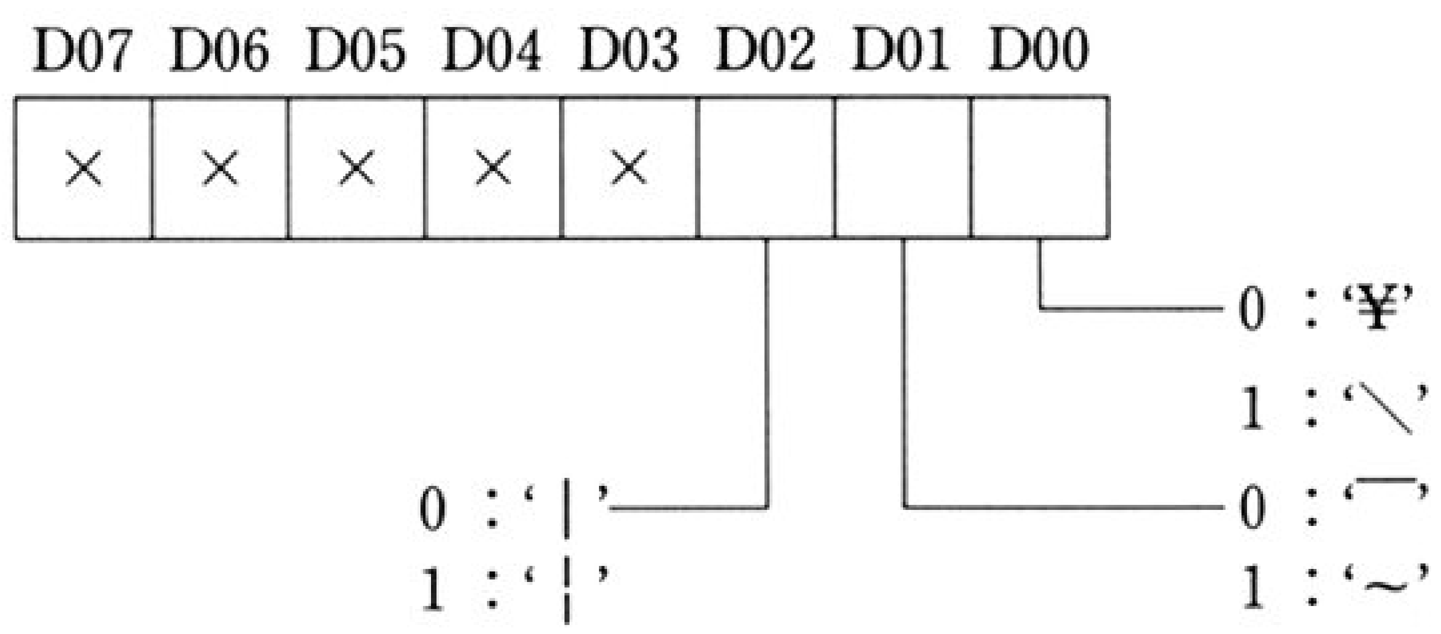
5　SRAMの内容

アドレス	サイズ、初期値	意　　味
$ED0028	1　14	コントラストのシステム既定値ワーク(0から15までの値)。
$ED0029	1　0	本体OFFのとき、フロッピーディスクをイジェクトするかどうかのフラグ。 0：イジェクトしません 1：イジェクトします
$ED002A	1　$0D	本体OFFのとき、テレビをコントロールするコード用ワーク。 $0D　：テレビOFF $05　：テレビモード そのほか、$01～$3Fのコードが指定可能です。
$ED002B	1　0	キーボードの配列を指定するフラグ 0：JIS配列 1：あいうえお配列
$ED002C	1　0	電卓の文字フォントを指定 0：LCDフォント 1：ノーマルフォント
$ED002D	1　0	SRAM($ED0100～$ED3FFF)の使用モード。 0：何も使用していません 1：SRAMDISKとして使用(SRAMDISK. SYSはこのモードでないと使用できません) 2：プログラムメモリとして使用
$ED002E	2　0	テキストパレット0のシステム既定値
$ED0030	2　$F83E	テキストパレット1のシステム既定値
$ED0032	2　$FFC0	テキストパレット2のシステム既定値
$ED0034	2　$FFFE	テキストパレット3のシステム既定値
$ED0036	2　$DE6C	テキストパレット4～7のシステム既定値
$ED0038	2　$4022	テキストパレット8～15のシステム既定値

5　SRAMの内容

アドレス	サイズ、初期値	意　　　味
$ED003A	1　3	キーのオートリピートが始まるまでの間隔用ワーク。 0～15 の値で指定(200ms＋100×Nms)。 3：(500ms＝200＋100×3)
$ED003B	1　2	キーのリピート間隔用ワーク。 0～15 の値で指定(30ms＋5×N^2ms)。 2：(50ms＝30＋5×2^2)
$ED003C	4　$80000	プリンタタイムアウト時間指定
$ED0040	4　0	SRAM が初期化されたときから前の回までの積算稼働時間(分)。 本体 OFF ごとに、前の回の稼働時間が積算されます。 リセットしたときも積算されます。
$ED0044	4　0	SRAM が初期化されたときから何回本体が OFF されたかの回数。 本体 OFF ごとに加算されます。
$ED0048～$ED0058	システム予約	
$ED0059	1　0	('¥', '‾', '|')の 3 文字を('＼', '~', '¦')に変換するフラグ(下図参照)。
$ED005A	1　1	ハードディスクが何台つながっているかのスイッチ。 (0～15)まで可能で、指定台数以上は IOCS コールのレベルでエラーになります。

$ED0059 の下図（ビット割り当て）：



5　SRAMの内容

アドレス	サイズ、初期値	意　　　味
		初期値は1ですから、2台以上使用する場合は書き直してください。
$ED005B～$ED00FF		システム予約
$ED0100		SRAMにプログラムを書くときの先頭アドレス
$ED0400		SRAMDISKの先頭アドレス
$ED3FFF		SRAMの終わり

注：SRAMの内容はシステムで使用していますので、書き替えないでください。

6 フロッピディスクコントローラ

X68000には、5.25インチ・2HD（両面高密度）タイプのFDD（フロッピーディスクドライブ）を2基内蔵しています。このFDDと拡張用のFDDコネクタに接続する拡張FDD（2基まで）を制御するフロッピーディスク・ドライブ・コントローラ（FDC）として、μPD72065を使用しています。

6.1 FDDの仕様

X68000のFDDは、次のような独自の機能を備えています。

(1)オートイジェクト機能

　指定したドライブに挿入されているフロッピーディスクを自動的に排出する機能

(2)アクセス表示LED点滅機能（2色LED）

　指定したドライブのアクセス表示LEDを点滅させる機能

FDD用のLEDの機能

パワー	アクセス表示LED	イジェクトLED
ON状態	・メディアがFDDに挿入されている場合、緑色が点灯 ・メディアがFDDに挿入されていないときでかつLED点滅機能がONの場合、緑色が点滅 ・メディアがFDDに挿入されていないときでかつLED点滅機能がOFFの場合、消灯 ・FDDがリード／ライト中緑色から赤色点灯に変化	・イジェクトマスク機能がONの場合、消灯 ・メディアがFDDに挿入されていてかつイジェクトマスク機能がOFFの場合、緑色が点灯
OFF状態	・メディアがFDDに挿入されている場合、緑色が点灯 ・メディアがFDDに挿入されていない場合、消灯	・メディアがFDDに挿入されている場合、緑色が点灯 ・メディアがFDDに挿入されていない場合、消灯

(3)イジェクトスイッチ禁止機能

　ディスクへ書き込み中などにイジェクトができないように、イジェクト機能を禁止する機能

(4)メディアの挿入／非挿入検出機能

　各ドライブへメディアが挿入されているか、いないかを常に監視する機能

6 フロッピーディスクコントローラ

(5)メディアの誤挿入検出機能

各ドライブのメディアの誤挿入を検出する機能

(6)割り込み機能

メディアを挿入／排出したときに割り込みを発生させる機能

内蔵 FDD の仕様

記憶容量 (KBytes)	アンフォーマット時	1667	
	フォーマット時	1065	IBM 準拠 26 セクター、256 バイト／高密度モード
	トラック容量	10.42	
データ転送速度 (kbits／sec)		500	
アクセスタイム (msec)	トラック間移動時間	3	シーク時の待ち時間＝ トラック間移動時間＋シークセトリング時間 平均トラック移動時間＝ 平均トラック移動時間＋ シークセトリング時間
	シークセトリング時間	15	
	平均アクセス時間	360	
メディア回転数 (rpm)		360	
スピンドルモーター起動時間 (sec)		0.5	
最内周記録密度 (BPI)		9870	
トラック数	TRACK／SIDE	77	
	TRACK／DISK	154	
トラック密度 (TPI)		96	
ヘッド数		2	
変調方式		MFM	FM 方式も可能
外形寸法		27.0(H)×148.0(W)×198.0(D):mm	
重量		900g	
その他		オートクランプイジェクト機構、オートリキャリブレート機能、VFO(SED9420AC)	

6.2 FDC（μPD72065）の特長

X68000では、フロッピーディスクを制御する FDC として μPD72065 を使用しています。この FDC の特長として次のものがあります。

- 同時に4基のFDDを接続が可能
- セクタサイズが可変できる（128，256，512，1024バイト／セクター）
- DMA転送が可能
- FDCに対して数十バイトのデータを与えるだけで、フロッピーディスクのフォーマットが可能
- エラー発生時に割り込みがかかり、FDCのコマンドが終了する
- ステップ速度やヘッドロード時間、アンヘッドロード時間は接続するFDDに合わせて変更が可能

6.3 FDCのリザルトステータス

リザルトステータスとは、CPUから与えたコマンドをFDCが実行したとき、発生したエラーなどを表すものです。これにはST0〜3まであり、それぞれ1バイトで構成されています。

・リザルトステータス0（ST0）

ビット7〜6：割り込み発生要因を示す
　00：コマンドの正常終了
　01：コマンドの異常終了
　10：無効なコマンドを実行しようとした
　11：ドライブ状態が変化した
ビット　5：シーク動作が正常終了または異常終了した時にセットされる
ビット　4：ドライブからFault信号を受けたときなどにセットされる
ビット　3：ドライブがReady状態でないときにセットされる
ビット　2：割り込み発生時のヘッドの状態を示す
ビット1〜0：割り込み発生時のドライブの番号を示す
　00：ドライブ0
　01：ドライブ1
　10：ドライブ2
　11：ドライブ3

・リザルトステータス（ST1）

ビット7：最終セクタを越えてアクセスを続けようとした場合セットされる
ビット6：常に0
ビット5：ディスク上のIDまたはCRCエラーが検出されたときにセットされる
ビット4：データ転送時にCPUまたはDMAが、FDCに対して規定時間内に入出力をできなかったときにセットされる

6　フロッピーディスクコントローラ

ビット3 ： 常に0
ビット2 ： 指定したセクタが見つからなかった時にセットされる
ビット1 ： ディスクは書き込もうとしたが、ライトプロテクトシールされていた場合にセットされる
ビット0 ： 未フォーマットなどでセクタが存在しない場合や、マークのみでデータフィールドがないときにセットされる

・リザルトステータス (ST2)
ビット7 ： 常に0
ビット6 ： 削除データ読み込み時に、通常データを読み込もうとした場合または、この逆の場合にセットされる
ビット5 ： CRC エラーが検出されたときにセットされる
ビット4 ： 指定したトラックが見つからなかった場合にセットされる
このビットがセットされてかつ、リザルトステータス1のビット2がセットされた場合はトラックが見つからなかったことが原因
ビット3 ： ベリファイコマンドでデータが一致した場合にセットされる
ビット2 ： ベリファイコマンドでデータが不一致であった場合にセットされる
ビット1 ： 指定したトラックの番号が既定外の場合にセットされる
ビット0 ： リザルトステータス1のビット0がセットされた原因が、マークのみでデータフィールドがない場合にセットされる

・リザルトステータス3 (ST3)
ビット7 ： Fault 信号の状態を示す
ビット6 ： Write Protected 信号の状態を示す
ビット5 ： Ready 信号の状態を示す
ビット4 ： Track0 信号の状態を示す
ビット3 ： Tow Side 信号の状態を示す
ビット2 ： Side Select 信号の状態を示す
ビット1 ： Unit Select1 信号の状態を示す
ビット0 ： Unit Select0 信号の状態を示す

7　IOCS. X

IOCS. X は、画面出力関係の IOCS を強化しデバイスドライバまたは常駐プログラムとして Human68k に組み込むことが可能です。
これにより、従来の IOCS コールを使っていたプログラムを書き換えることなく、高速な画面出力を実現できます。
また、エスケープシーケンスも拡張されています。

7.1　IOCS. X の組み込み／切り離し

IOCS. X を Human68k に組み込むには、2 つの方法があります。

(1) デバイスドライバとして組み込む場合
CONFIG. SYS のなかで

```
DEVICE = IOCS. X 〈スイッチ〉
```

という 1 行を記述することで、Human68k のデバイスドライバとして利用ができます。
また、次のスイッチを指定することが可能です。

／D：コンソール出力を高速化しない
IOCS.X での拡張機能は利用できますが、画面出力の高速化は行いません。

(2) 常駐プログラムとして組み込む場合
Human68k のコマンドラインから、次のように入力します。

```
A>IOCS ⏎
```

これで、拡張された IOCS コールが利用可能になります。
このときに (1) と同様に／D のスイッチを指定することも可能です。
(スイッチの指定は、'—', '／'の両方が使用できます。)

IOCS. X を切り離し ROM 版 IOCS に切り替えるには、Human68k のコマンドラインから、次のように入力します。

```
A>IOCS　−R ⏎
```

7　IOCS. X

これで、IOCS. X は切り離され ROM 版の IOCS コールの利用ができます。
IOCS. X を常駐プログラムとして組み込んだ場合は、切り離しと同時にメモリも開放します。
デバイスドライバとして組み込んだ場合は、IOCS. X の切り離しだけを行いメモリの開放は行いません。
また、デバイスドライバとして IOCS. X を組み込んである場合には、切り離し後コマンドラインから

```
A>IOCS ↵
```

と入力することで IOCS. X を再度、組み込むことが可能です。

(3) ショートカットで IOCS. X を呼ぶ場合

IOCS コール$00～$FF の各処理アドレスはメモリの$400～$7FF に格納されています。
例えば、IOCS B_KEYINP を実行したいとき通常は、

```
moveq   #0, d0
trap    #15
```

としますが、処理を速くするため

```
movea. l $400+40, a0
jsr      (a0)
```

としても同じ結果が得られます。
この場合、必ず A5 レジスタを 0 にしてください。
これは、拡張された IOCS コールの新しいサブルーチン群が、IOCS ワークメモリの内容を A5 の値をもとにして参照しているためです。

7.2 拡張された IOCS コール

次に機能が拡張された IOCS を個々に説明します。
ここでは ROM 版 IOCS コールとの相違点のみを記載していますので、従来の機能については本書の第 3 章を参照してください。

7　IOCS. X

■ B_CONSOL　　IOCS コール番号：$2E　（機能追加）

機能　表示範囲指定

入力　D0. L　$2E　IOCS コール番号
　　　D1. L　表示開始ドット位置
　　　D2. L　表示範囲指定

解説　ROM 版 IOCS では、横方向の表示開始ドット位置が 16 ドット単位（16 の倍数）でしたが、8 ドット単位（8 の倍数）で指定できるようになりました。
ただし、8 ドット単位の指定を ROM 版 IOCS で行うと、アドレスエラーが出ます。
表示領域だけスクロールしたい時は、B_CONMOD（$AD）のモード 18、ソフトコピースクロールを指定してください。
このコールを実行後カーソルは、ホーム位置に移動します。

■ B_CONMOD　　IOCS コール番号：$AD　（新規 IOCS コール）

機能　カーソルおよびスクロールのコントロール

入力　D0. L　$AD　　IOCS コール番号
　　　D1. L　MODE　　カーソルモード
　　　D2. L　ADDRESS　　スクロールモード

解説　D1. L で指定されたカーソル（スクロール）のコントロールをします。
D1. L ＝ 1……カーソルの点滅をする（既定値）
D1. L ＝ 2……カーソルの点滅を禁止する
D1. L ＝ 3……カーソルのパターンを指定する
　　レジスタ D2. L の値により、動作が次のようになります。
D2. L の下位ワードが 0：内部バッファに定義されたカーソルパターンをカーソルとして描画します。
D2. L の下位ワードが 0 以外：下位バイトをプレーン 0、上位バイトをプレーン 1 の排他論理（EXOR）のデータとし、D2 の上位ワードで指定されるスタートラインからカーソルを描画します。スタートラインは 0～15 で指定します。

D2. Lが$0000FFFFであれば、今までのカーソルと同じになります。

D1. L ＝ 4……D2. Lで示されるアドレスのカーソルパターンデータを内部バッファにコピーします。
カーソルパターンは、最初の16バイトがプレーン0、次の16バイトが、プレーン1のパターンとなります。

D1. L ＝ 16……スムーススクロールの指定
D2. L ＝ 0……ジャンプスクロール（既定値）
D2. L ＝ 1……4ドットスクロール
D2. L ＝ 2……8ドットスクロール
D2. L ＝ 3……16ドットスクロール

注意：スムーススクロールを使う場合、文字を連続的に出力しなければ、スクロールのときに画面が乱れます。
垂直同期（16ms）の間に、1ライン出力を終了してください。

D1. L ＝ 17……ラスターコピースクロールの指定（既定値）
D1. L ＝ 18……ソフトコピースクロールの指定

注意：広範囲でのソフトスクロールは、表示速度が遅くなります。

■ DRAWMODE　　IOCSコール番号：$B0　（新規IOCSコール）

機能　グラフィック描画モードの指定

入力　D0. L ＝ $B0　　IOCSコール番号
D1. L ＝ MODE　描画モード

リターン　前のモード

解説　グラフィック描画の際の描画モードを変えます。

D1. L ＝ 0……通常のモード
D1. L ＝ 1……反転モード

反転モードで描画した場合、描画するグラフィックデータの色と、そのデータが描画される場所にあらかじめ塗られている色の否定（NOT）を計算し、その計算結果の色でグラフィックデータを描画します。
ラバーバンドのラインや、十字カーソルなど、一時的に画面にラインを描画したいときに使います。
このモードにして描画ルーチンを2回呼ぶと、画面データが元に戻ります。
この機能を使い終えたら、必ず通常の描画モードに戻してください。
D1. L に−1を指定した場合、現在のモードを返します。

■ TXLINE　　IOCS コール番号：$D5（新規 IOCS コール）

機能　テキスト画面のライン

入力　D0. L　$D5　　IOCS コール番号
A1. L　パラメータのポインタ

リターン　なし

解説　テキスト画面のライン描画を行います。
ラインと重なった文字は、消去されます。

0(A1). W　テキストプレーン番号
2(A1). W　始点 X 座標
4(A1). W　始点 Y 座標
6(A1). W　終点 X 座標
8(A1). W　終点 Y 座標
10(A1). W ラインスタイル

注意：CRTC（CRT コントローラ）のビットマスク機能を使っていますので、マウスカーソルは OFF にして、このコールを呼び出してください。

7.3　拡張されたエスケープシーケンス

IOCS. X を組み込むことで、従来のエスケープシーケンスに加え下表のエスケープシーケンスが利用できます。
従来のエスケープシーケンスについては、本体添付の「Human68k ユーザーズマニュアル」の付録を参照してください。

シーケンス	機　能
ESC [?4h	スムーススクロールにします。
ESC [?4l	ジャンプスクロールにします。
ESC [pnP	カーソルの位置から pn 文字削除します。 カーソル位置は変更ありません。
ESC [pn @	カーソル位置にpn個の空白文字を属性オフの状態で挿入します。 カーソル位置は変更ありません。
ESC [pnX	カーソルの位置から pn 文字を属性オフにしてクリアします。 カーソル位置は変更ありません。

索引 50音順

あ

い

う

え

お

か

き

く

こ

さ

し

す

せ

そ

た

ち

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

よ

ら

り

アルファベット順

シャープ株式会社

本　　社　〒545 大阪市阿倍野区長池町22番22号
電子機器事業本部　〒329-21 栃木県矢板市早川町174番地
ＡＶＣシステム事業推進室
〒162 東京都新宿区市谷八幡町８番地　電話（03）3260-1161（大代表）
東京支社内　電子機器事業本部　ＡＶＣシステム事業推進室　ソフトウェア担当